实验 - 二进制基础

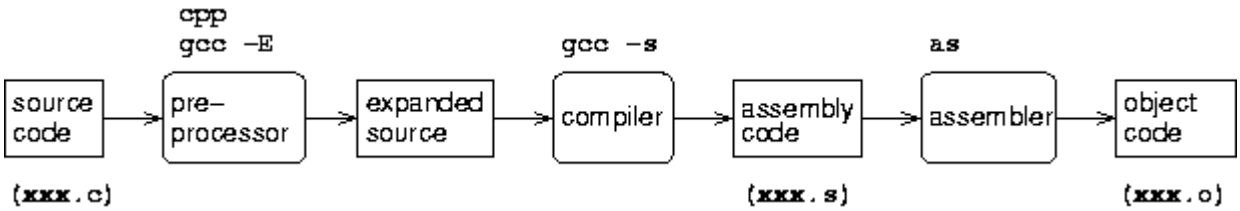
实验简介

在第二节中，我们学习了程序的编译、链接和装载，细节到了ELF的文件头，布局，静态链接和动态链接等等。此次实验重在回顾课堂上的内容，并就课堂最后涉及到的程序运行过程（`_start`）进行实验。此次作业完成后，将基本掌握二进制程序分析工具 `readelf, objdump`

基础部分

1. 在 linux 上编译、链接、装载程序 (40分)

我们首先复习一下这三个过程，对于编译，即从源代码到目标代码其实还能更细分如下步骤



1. 预处理：`gcc -E` 完成宏的展开
2. 编译：`gcc -S` 完成从源码到汇编的转化
3. 汇编：`as` 从汇编代码到架构相关的机器码

> 其中，编译的步骤往往又会划分为前端和后端，当然，这些细节讲到高年级的编译原理课程中再深入学习

准备源代码

实验要求学生准备两份C源代码，一份是如下的，简单的 hello world，我们称之为 `a.c`

```c
#include <stdio.h>
int main()
{
  printf("Hello World!");
  return 0;
}
```

另外一份，请学生自行编写一份C代码，或者从之前的C程序设计中拿一份代码来用即可，称之为 `b.c`

> 对于在Linux环境上如何编程，你可以在 host 机器编写好后上传到 guest，或者学习终端工具如 vim，或者桌面工具 gedit

预处理

我们通过如下指令对源代码进行预处理

```
gcc -E a.c -o a.c.txt
gcc -E b.c -o b.c.txt
```

查看生成的输出文件, 比较输出文件和原始文件的区别.

编译

我们通过如下指令对源代码进行编译

```
gcc -S a.c -o a.s
gcc -S b.c -o b.s
```

查看生成的输出文件, 比较输出文件和原始文件的区别.

汇编

我们通过如下指令对源代码进行汇编

```
as a.s -o a.out
as b.s -o b.out
```

用`objdump -d`查看生成的输出文件, 比较输出文件和.s文件区别.

问题

- 尝试运行汇编生成后的文件, 为什么它们不能被直接运行?

- 使用gcc -v -static 编译一个c文件, 截图标识出其中的编译命令, 汇编命令和链接命令.

2. 分析 ELF 文件头 (30分)

> 注，下面写程序不限使用语言，C、python均可，但不能使用已有的工具包或者库代码

ELF 文件头位于目标文件最开始的位置，含有整个文件的一些基本信息。文件头中含有整个文件的结构信息，包括一些控制单元的大小。

可以使用下面的数据结构来描述 ELF 文件的文件头(64位)。

```
#define EI_NIDENT (16)

typedef struct
{
  unsigned char e_ident[EI_NIDENT]; /* Magic number and other info */       // 1 byte * 16
  Elf64_Half    e_type;         /* Object file type */                  // 2 bytes
  Elf64_Half    e_machine;      /* Architecture */
  Elf64_Word    e_version;      /* Object file version */               // 4 bytes
  Elf64_Addr    e_entry;        /* Entry point virtual address */       // 8 bytes
  Elf64_Off e_phoff;        /* Program header table file offset */  // 8 bytes
  Elf64_Off e_shoff;        /* Section header table file offset */
```

```
  Elf64_Word    e_flags;        /* Processor-specific flags */
  Elf64_Half    e_ehsize;       /* ELF header size in bytes */
  Elf64_Half    e_phentsize;        /* Program header table entry size */
  Elf64_Half    e_phnum;        /* Program header table entry count */
  Elf64_Half    e_shentsize;        /* Section header table entry size */
  Elf64_Half    e_shnum;        /* Section header table entry count */
  Elf64_Half    e_shstrndx;     /* Section header string table index */
} Elf64_Ehdr;
```

使用`readelf -h a`查看elf文件的文件头信息, `hexdump a`查看文件原始的16进制信息, 将这两个信息和上面这个数据结构一一对应起来. 写一个程序实现类似于`readelf -h`的功能.

> - 不需要和readelf -h 完全一致, 以e_type为例, 输出可以只写到 Type: 1, 即读取出文件头中对应数据结构的数字, 不需进行进一步的解析(但如果好奇,可以参考这里).
> - 把二进制数据解析成数字的过程需要知道文件是大端序还是小端序, 即e_ident[5], 该值为1则是小端序, 值为2则是大端序
> - 根据e_ident[5]的值对之后文件头的数据进行解析(关于大端序和小端序的解释,没学过汇编的同学可以看这里)

```
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:               0x401000
  Start of program headers:          64 (bytes into file)
  Start of section headers:          12656 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         5
  Size of section headers:           64 (bytes)
  Number of section headers:         8
  Section header string table index: 7
```

3. 分析 ELF 各段结构 (30分)

在 ELF 文件中可以包含很多**节**（Section），所有这些节都登记在一张称为**节头表**（Section Header Table）的数组里。节头表的每一个表项是一个`Elf64_Shdr`结构，通过每一个表项可以定位到对应的节。

> - 也需要读取文件头中的e_ident[5]字段, 确定以小端序还是大端序解析文件
> - 也可以只解析到数字, 不用进一步解析

```
typedef struct
{
  Elf64_Word    sh_name;     /* Section name (string tbl index) */    // 4 bytes
  Elf64_Word    sh_type;     /* Section type */
  Elf64_Xword   sh_flags;    /* Section flags */                      // 8 bytes
  Elf64_Addr    sh_addr;     /* Section virtual addr at execution */  // 8 bytes
  Elf64_Off sh_offset;   /* Section file offset */                // 8 bytes
  Elf64_Xword   sh_size;     /* Section size in bytes */
  Elf64_Word    sh_link;     /* Link to another section */
  Elf64_Word    sh_info;     /* Additional section information */
  Elf64_Xword   sh_addralign;    /* Section alignment */
  Elf64_Xword   sh_entsize;  /* Entry size if section holds table */
} Elf64_Shdr;
```

写一个程序实现类似于`readelf -S`的功能.

挑战部分

- 使用 makefile 编译helloworld程序

https://www.oreilly.com/library/view/c-cookbook/0596007612/ch01s16.html

- 理解make 工具与直接编译器的区别