1. 概述

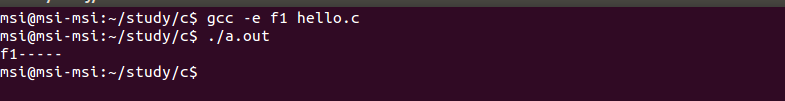
众所周知，无论是运行在操作系统上的应用程序的编译，还是操作系统本身的编译（包括bootloader），都会涉及到链接问题。

对于应用程序而言，链接就是将目标文件组合成一种特定的格式，如ELF，PE等等，当执行这种特殊格式的文件时，操作系统的加载器会去解析格式，并根据解析得到的信息找到真正需要执行的代码。而对于操作系统本身，或者bootloader，则需要通过链接将庞大的目标文件合理的组织在一起，这里所谓合理大部分时间指的是目标文件中的段（SECTION）在内存中的分配，以及各个段的排布和入口地址（也就是第一条执行的指令）。

但不管是应用程序的链接，还是bootloader的链接，对于链接过程的控制都分成两种：

1. **命令行**

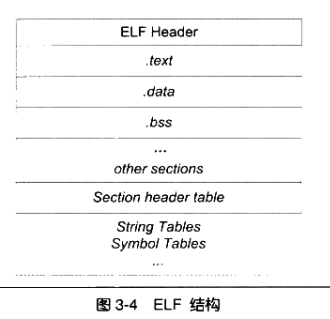
比如下图中，通过gcc –e参数，我们可以指定a.out的入口地址为函数f1



1. **链接脚本**

如我们将要学习的uboot链接脚本：u-boot.lds(只截取了前面一部分)

1. OUTPUT\_FORMAT("elf32-littlearm", "elf32-littlearm", "elf32-littlearm")
2. OUTPUT\_ARCH(arm)
3. ENTRY(\_stext)
4. SECTIONS
5. {
6. . = 0x00000000;
8. . = ALIGN(4);
9. .text :
10. {
11. \*(.\_\_image\_copy\_start)
12. SDIR/start.o (.text\*)
13. \*(.text\*)
14. }
15. ELF文件
    1. ELF文件结构



我们生成的.o文件，a.out文件，以及u-boot，它们均是按照上面的结构图来组织，根据上面的结构图，我们可以得到入口地址，已经整个目标文件（或者可执行）的符号等等，这些信息有重要的意义，它告诉我们整个密密麻麻的二进制数字是怎么组织的，以及我们的代码该从哪里以及哪一句开始！

1. GCC链接脚本语法
2. S5P6818:u-boot.lds分析