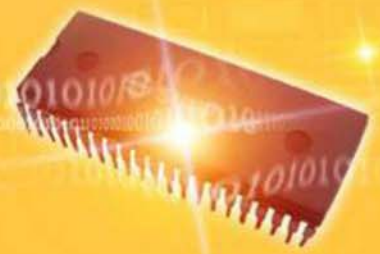
嵌入式系统工程师



裸机工程开发调试

大纲

- GNU常用工具的使用
- 程序的编译链接过程
- 裸机工程构建及调试

裸机工程开发调试

- GNU常用工具的使用
- 程序的编译链接过程
- 裸机工程构建及调试

GNU常用工具的使用

- GNU组织不仅给我们带来了许多开源软件工程，还带来了强大的GNU编译工具
- GNU提供的常用工具包括:
  - 预处理器　　cpp
  - C编译器　　gcc
  - C++编译器　　g++
  - 汇编器　　as
  - 链接器　　ld
  - 二进制工具集 objcopy、objdump、……
- ls /usr/local/arm/4.3.2/bin/
- 下面介绍几个常用的工具:

GNU常用工具的使用

- nm: 符号显示器
  - 显示符号 $nm -n main_elf
  - 显示内容:
    - 第一列为符号地址
    - 第二列为符号所在段
    - 第三列为符号名称
  - 如下页图所示

GNU常用工具的使用

```
@sunplusedu$
@sunplusedu$pwd
/usr/local/arm/4.3.2/arm-none-linux-gnueabi/libc/usr/lib
@sunplusedu$arm-linux-nm -n crt1.o
         U __libc_csu_fini
         U __libc_csu_init
         U __libc_start_main
         U abort
         U main
00000000 R _IO_stdin_used
00000000 D __data_start
00000000 T _start
00000000 W data_start
@sunplusedu$
```

- 各段含义见下页图:

GNU常用工具的使用

段	描述
b/B	.bss(b静态/B非静态)未初始化变量
d/D	.data(d静态/D非静态)已初始化变量
r/R	.rodata(r静态/R非静态)只读数据段
t/T	.text(t静态/T非静态)函数
A	不可改变的绝对值
C	.o中未初始化非静态变量
N	调试用的符号
U	表示符号只有声明没有定义

注意：灰色阴影部分只需了解

GNU常用工具的使用

- nm符号显示器总结:
  - 静态变量和非静态的全局变量，所分配的段只与其是否初始化有关，如果初始化了则被分配在.data段中，否则在.bss段中
  - 函数无论是静态还是非静态的，总是被分配在.text中，小写t表示静态，大写T表示非静态
  - 函数内的局部变量由于是分配在栈上的，所以在nm中是看不到他们的

GNU常用工具的使用

- objdump: 信息查看器
  - 查看所有段信息 $objdump -h main_elf
  - 查看文件头信息 $objdump -f main_elf
  - 查看反汇编 $objdump -d main_elf
  - 查看内嵌反汇编 $objdump -S -d main_elf
- objcopy: 段剪辑器
  - 去除elf格式信息

    $objcopy -O binary -S main_elf main.bin

裸机工程开发调试

- GNU常用工具的使用
- 程序的编译链接过程
- 裸机工程构建及调试

程序的编译链接过程

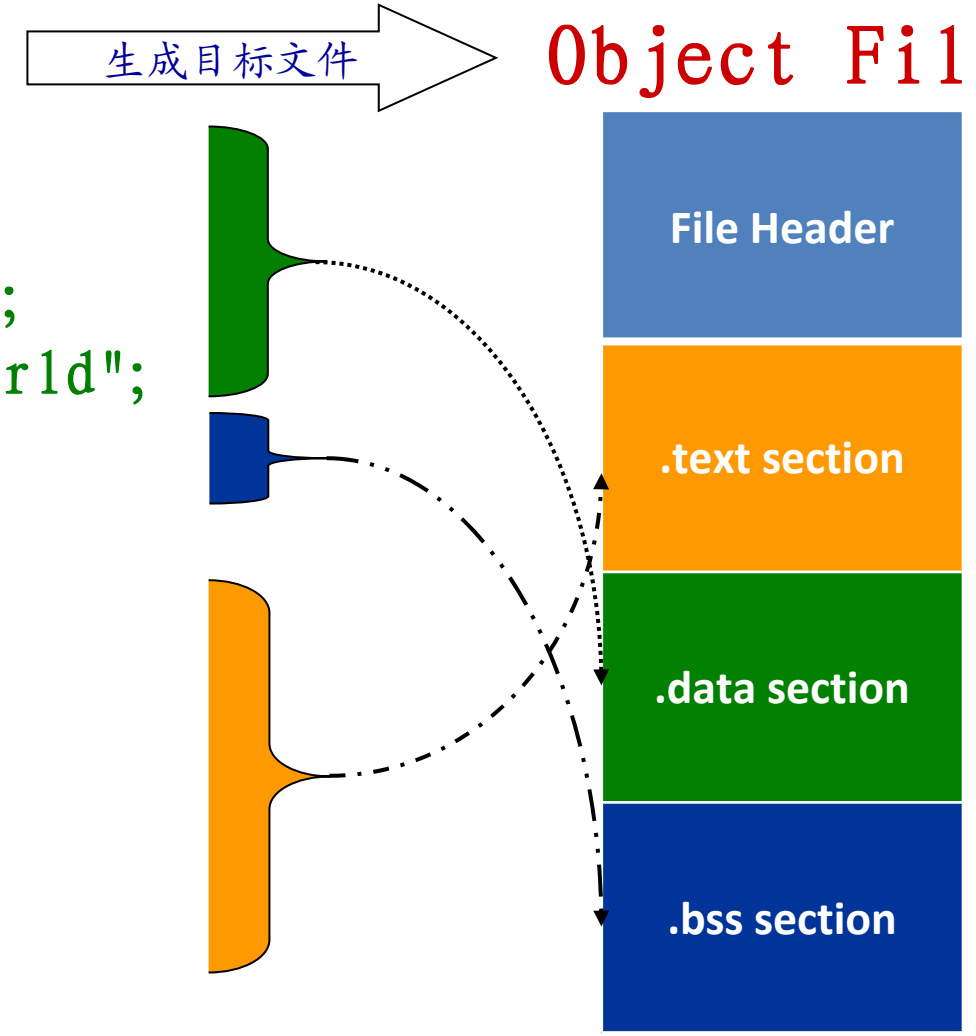
- 回顾一下应用程序的编译链接过程?
  - gcc -E -o main.i main.c
  - gcc -S -o main.S main.i
  - gcc -c -o main.o main.S
  - gcc -o main_elf main.o
- 最后一步是将目标文件交给链接器链接生成可执行文件
- 什么是链接呢?

程序的编译链接过程

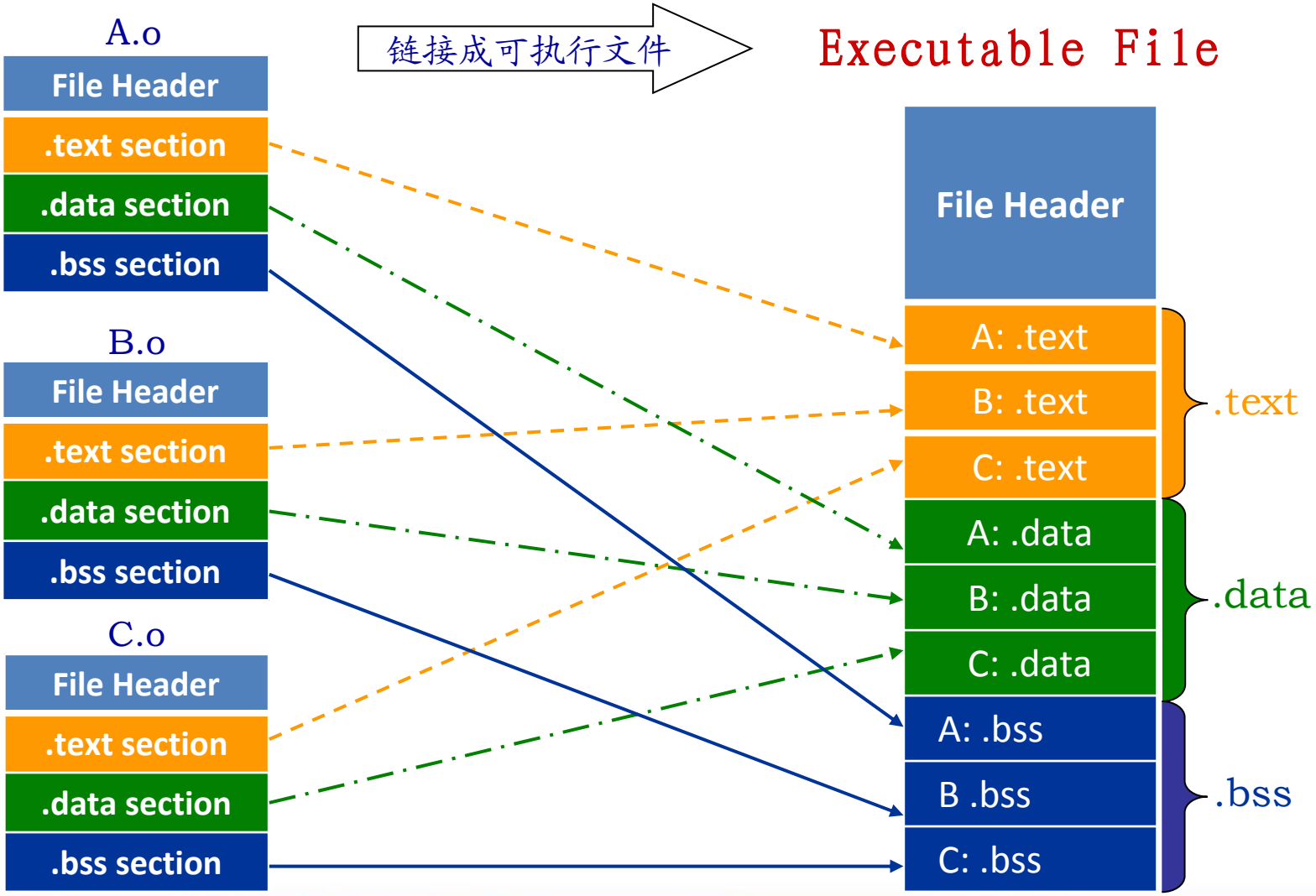
C code

```
int b = 0x12;
const int c = 0x34;
static int d = 0x56;
char *p = "hello world";
int a;

int main(void)
{
    b++;
    a = 0;
    return 0;
}
```



程序的编译链接过程



程序的编译链接过程

- 用户通常并不直接参与链接过程，那么可执行程序是如何链接生成的呢？
- 一个标准应用程序通常都由链接器采用默认链接脚本(arm-linux-ld --verbose)将“用户程序”和“库”共同链接生成可执行程序
- 假如不采用系统给定的默认链接脚本，我们应该怎样进行链接？
  - ld -o main main.o

程序的编译链接过程

- 结论:
  - 链接过程中需要一个标号(_start)作为程序入口
  - 标号(_start)的作用是: 将用户程序从汇编带到了C语言程序入口，即main()函数，从此开始我们的应用程序之旅
  - _start标号所在的汇编文件在编译工具链下面: 4.3.2/arm-none-linux-gnueabi/libc/usr/lib/crt1.o
- 如果没有crt1.o等标准库文件，我们应该怎样完成可执行文件的生成?

程序的编译链接过程

- 我们可以自己实现_start入口，或者重新指定一个新的入口

- 手动链接并生成可执行文件过程如下：
  - 直接通过参数指定程序入口和段地址：

```
arm-linux-ld -Ttext=0x30000 -Tdata=0x40000
-e main -o app head.o main.o
```

  - 通过链接脚本来指定程序入口和段地址：

```
arm-linux-ld -Tapp.lds -o app head.o main.o
```

程序的编译链接过程

```
//连接文件app.lds为:
ENTRY(main)
SECTIONS
{//“*”号指所有目标，可以指定.o目标文件，多个用空格隔开
        .=0x30000;//“.”指的是当前位置
        .text:{*(.text)}
        .=0x40000;
        .data:{*(.data)}
        .bss:{*(.bss)}
}
```

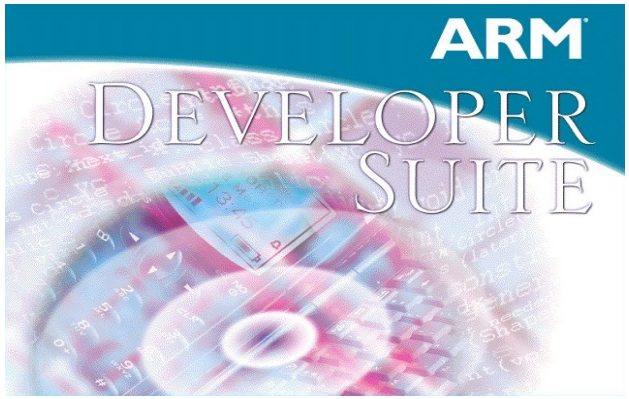
- 为什么需要我们自己完成链接过程？

裸机工程开发调试

- GNU常用工具的使用
- 程序的编译链接过程
- 裸机工程构建及调试

程序的编译链接过程

- 目前常用的ARM编译开发环境:



- 思考：能不能搭建自己的开发环境？

裸机工程构建及调试

- 裸机程序完整编译过程:
  - gcc -c -o main.o main.c
  - gcc -c -o uart.o uart.c
  - ld -Tapp.lds -o app_elf main.o uart.o
  - objcopy -O binary -S app_elf app.bin
  - 交叉编译时工具开头均加上: arm-linux-
- 裸机工程管理文件
  - 各目录Makefile
  - 链接脚本app.lds
  - 规则文件Rules.mk

裸机工程构建及调试

- 最后将app.bin下载到裸板内存中并执行
  - 工程实例：00-pro-demo
  - 下载命令：update app
  - 运行命令：go
  - 退出程序：crtl+c

凌阳教育
www.sunplusedu.com

值得信赖的教育品牌

Tel: 400-705-9680 , Email: edu@sunplusapp.com , BBS: bbs.sunplusedu.com