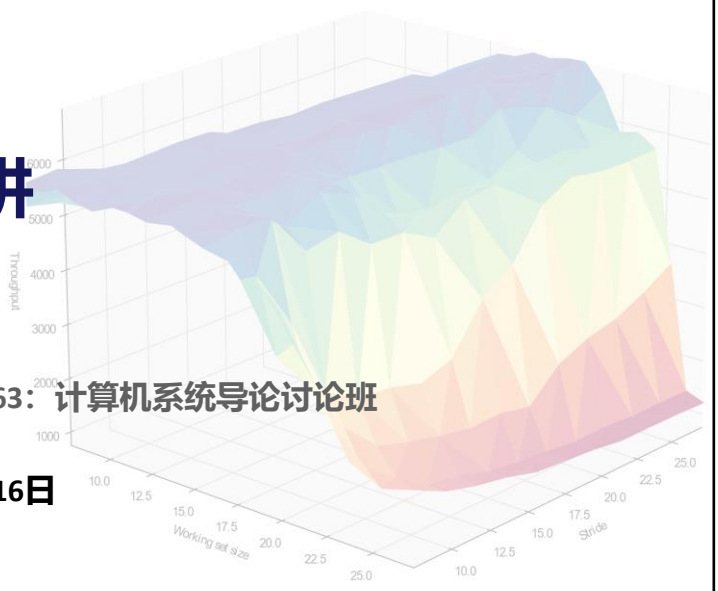
# 第六讲




PKU 04832363：计算机系统导论讨论班

王畅

2021年11月16日

## 重点细节

- P466底部，目标文件有三种，但在Unix下都是以ELF形式组织的。
- P467 ELF的典型格式（记忆）：.bss相关还有一个伪节COMMON，全局变量是否初始化放置位置有所不同。具体解释在P474中部。每个节的解说都要认真看一遍。
- P471符号解析只涉及全局符号定义。P473多个不同弱符号，任选一个，Linux下默认选最长类型的任意一个。
- P483-485 ELF加载过程易忽略，要读，例如程序入口点是什么，_init做什么，_start做什么。

2

BSS和COMMON的区别：只有未初始化的全局变量会进入COMMON，初始化为0的全局变量和静态变量、未初始化的静态变量都进入BSS。看书P469页中下部的规定。

当然，初始化为非0的全局变量和静态变量都进入DATA。

## 静态变量与符号表

- In computer programming, a static variable is a variable that has been allocated “statically”, meaning that its lifetime is the entire run of the program.
- 过程中定义的静态变量会出现在符号表中，但是会有多个版本，用随机数字来identify（书P468底部）。
- 但过程外定义的静态变量编译器不需要作区分（scope）。
- A function-local static variable is something different than a global static variable. Since there can be as many function-local statics with the same name as you like (provided they are all in different scopes), the compiler might have to change their names internally (incorporating the function's name or the line number or whatever), so that the linker can tell them apart.

3

因此在符号表中，静态变量有两种。第一种是过程中定义的静态变量，名字需要添加随机数字区分；第二种是过程外定义的静态变量，名字不需要添加随机数字区分。

## 重定位

- **为何要重定位?**
  - 对于一个目标文件中的全局变量、静态变量和全局变量，编译器可以确定它们在目标文件中的相对地址。
  - 但是在合并这些目标文件时，会发现可能在本文件中用到了其他源文件中定义的符号或者需要一个绝对地址，编译器对此无能为力，只能交给重定位填充最后确定的地址。
- **哪些需要重定位?**
  - 凡是进行外部函数调用的语句都要重定位。
  - 凡是用到非局部变量（含全局变量、静态变量）的语句也都要重定位。
  - 凡是初始化时，引用其他非局部变量的地址的非局部变量，都要重定位。

4

最后三句话非常重要，它决定了`.rel.text`和`.rel.data`中都出现什么。请谨记在心，后面会反复强调。（书`P467`页倒数两段的补充说明。）

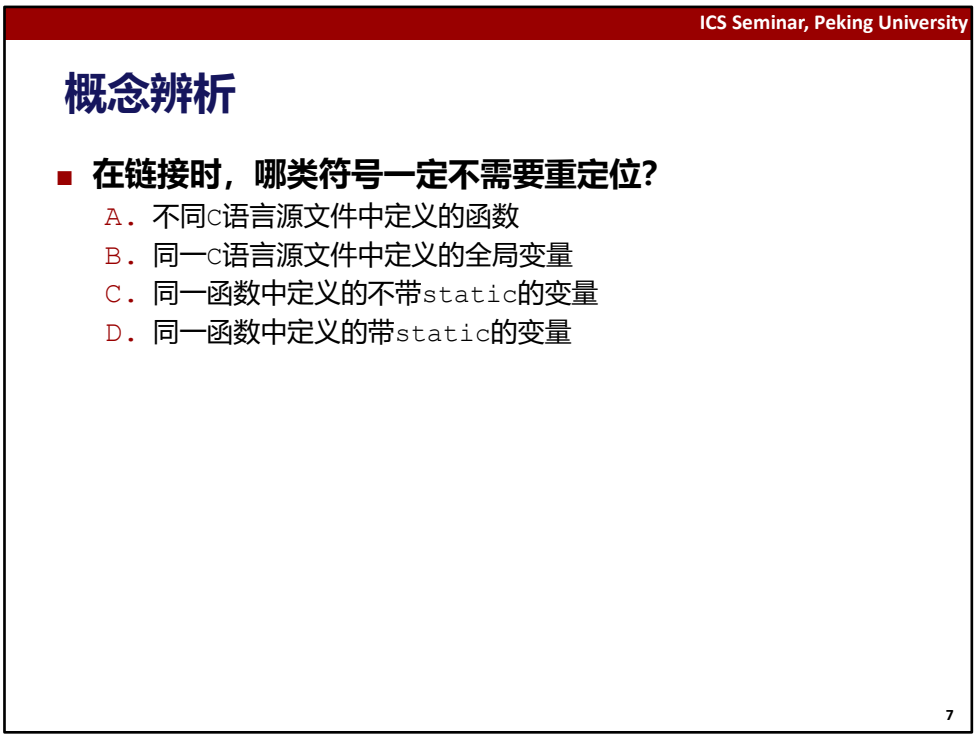
原理：调用函数时需要指明的是相对地址，因此同一个源文件中的调用不需要进行重定位。使用非局部变量时要用到绝对地址，因此只要用到了其地址就要重定位。其次，指令的重定位和数据初始化时的重定位是不一样的。请注意，非局部变量的初始化本质上不是一条指令。

Activity 1

# 轮流回答问题

# 书中练习

- 7.1
- 7.2
- 7.5



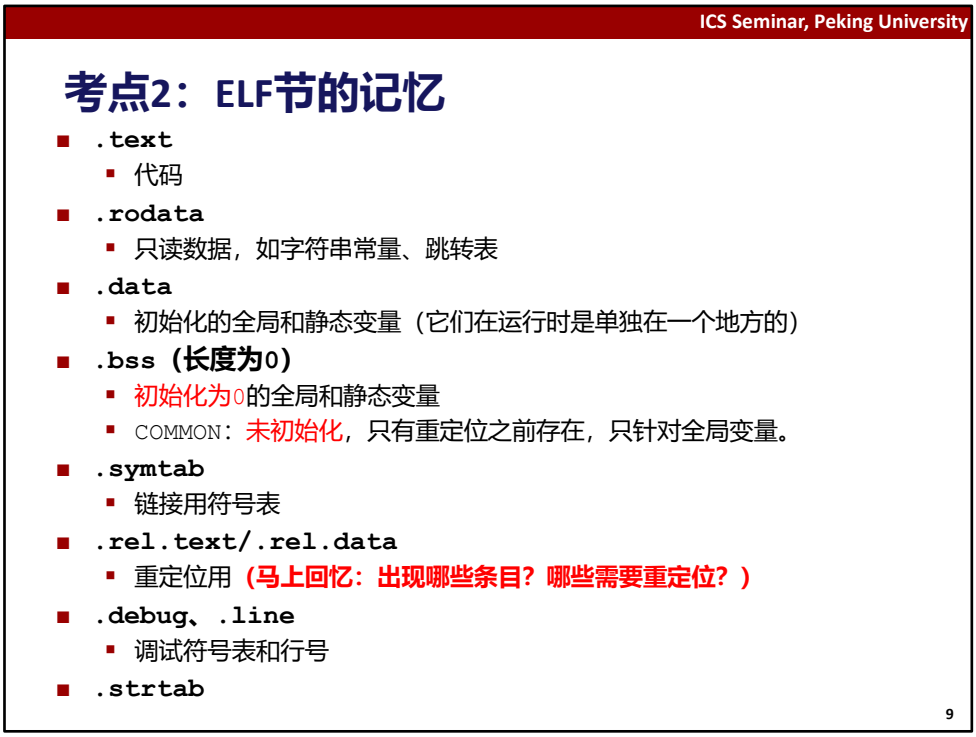

C，即局部变量。请注意这里说的是一定不需要，有一些非局部变量或者指令是可能需要重定位也可能不需要重定位的。参看“概念辨析”的最后一张投影片的题目分析。

## 概念辨析

- **下列关于链接技术的描述，错误的是：**
  - A. 在Linux系统中，对程序中全局符号的不当定义，会在链接时刻进行报告。
  - B. 在使用Linux的默认链接器时，如果有多个弱符号同名，那么会从这些弱符号中任选一个占用空间最大的符号。
  - C. 编译时打桩需要能够访问程序的源代码，链接时打桩需要能够访问程序的可重定位对象文件，运行时打桩只需要能访问可执行目标文件。
  - D. 链接器的两个主要任务是符号解析和重定位。符号解析将目标文件中的全局符号都绑定到唯一的定义，重定位确定每个符号的最终内存地址，并修改对那些目标的引用。

8

A，可能也不报告，例如多个弱符号任选一个，可能造成非预期行为。




BSS和COMMON的区别见前面注释。

Strtab, is the string table for all other references. When you read symbols from an ELF object, every Symbol structure (Elf32_Sym) has a member called st_name. This is an index into strtab to get the string name of that symbol.

根据前面讲到的哪些需要重定位的规律，我们可以确定.rel.****中会有什么东西。

.rel.text对应那些引用了非局部变量或者调用了外部函数（同一个源文件中的函数不算！）的指令。

.rel.data很稀少，只对应那些初始化时，使用了一个全局函数的地址或者非局部变量的地址的非局部变量。

## 概念辨析

- 在foo.c文件中包含如下代码：

```
int foo(void) {
    int error = printf("You ran into a problem!\n");
    return error;
}
```

- 经过编译和链接之后，字符串"You ran into a problem!"会出现在哪个段中?



.rodata

## 概念辨析

- 在foo.c文件中的函数以外，如果添加下面一条语句

```
static int count = 0xdeadbeef;
```

- 编译为foo.o得到的ELF可重定位目标文件中，哪些区域会发生变化?

11

.data, .symtab。

注意不要误解了.rel.data的含义，这里.rel.data是不会发生变化的，因为count初始化时，不需要引用别的非局部变量的地址，换言之它作为一个数据，初始化时不涉及重定位问题。别的位置引用它可能涉及重定位问题，但那是.rel.text的事情。另一方面，如果题目改为static int* count = &x;，其中x为一个全局变量，那么.rel.data是会发生改变的。或者，再加一条int y = count;，那么.rel.data也会改变，但这个对应的是y。

.rel.text也不会发生变化，除非增加一条引用count的指令，例如count = 3;，那么这条指令需要重定位，会在.rel.text中对应重定位条目。

原则上.strtab也会发生改变（但我们一般不考虑这个）。

Activity 2

# 问题求解

# 考点1：生成目标程序的全过程

- 理解生成过程是一个“流水线”。
- Section 11 练习1

## 考点3：结合汇编的符号解析/重定位分析

- **符号解析冲突解决**
  - 只针对全局符号（但局部符号在符号表中），记忆强弱符号选择规律，注意记住强弱符号放置的位置。
- **重定位分析**
  - 链接器会确定好每个定义和引用位置的绝对地址。因此需要**计算**相对位置或者**直接输入**绝对位置。其中绝对引用不用管offset（offset是指当前指令的位置）。最后要补addend，这有时只有在控制转移型指令时要用到（为什么？）。（不要死记硬背公式）
  - 如何读OBJDUMP得到的信息？
    - 例：a: R_X86_64_32 array，偏移a，绝对定位，无addend
    - 例：f: R_X86_64_PC32 sum-0x4，偏移f，相对定位，addend-4
- **Section 11 练习3、4**

14

Extern或者未定义的变量会出现在符号表中，但是无法谈强弱（因为是declaration而不是definition），如果一定要谈强弱就填弱。Extern的变量在本模块中属于的是UND伪节。

相对：计算当前引用的绝对地址（基址+offset），知道目标位置的绝对地址，计算相对地址：后者减前者，然后补上addend。

绝对：直接用目标的绝对地址，然后补上addend。

请注意这里的offset是需要修改引用的那个位置相对其所在section的开头的（这个位置可能在指令的中间！注意课本算法foreach section s）。也就是说，对于指令中的relocation，是用.text的偏移量，而对于初始化中的偏移量，则一般应该是用.data的偏移量（相对于那个被初始化符号所在section的开头）。

Addend的含义如下：An addend is simply something that should be added to the fixed up address to find the correct address. For example, if the relocation is for the symbol i because the original code is doing

something like i[8] the addend will be set to 8. This means “find the address of i, and go 8 past it”.

就实际而言，因为addend的储存和表示方式不同，重定位条目还可进一步分类（课本中给出的是常用的一种），但超出了课程范围。

注意最后填入指令中的重定位信息要遵循小端法!

# any questions?

Thanks & 感谢观看