第 6 次作业

1. 编译器在生成可重定位目标文件（.o）时，非静态全局变量、静态全局变量、非静态局部变量、静态局部变量、常量字符串（如 printf()的格式串）、函数，它们地址能确定吗？

答案：

只有非静态局部变量的地址能确定，其他符号都需要重定位。

2. 在需要重定位的符号中，哪些在链接的时候重定位？哪些需要在运行的时候重定位？

答案：

程序本身定义的符号以及静态链接（静态库）的外部符号，是在链接的时候重定位的。动态链接（动态共享库）的外部符号，需要在运行的时候重定位。

3. 可重定位目标文件（.o）一般都包含如下的节：

.text、.data、.rodata、.bss、.symtab、.strtab、.shstrtab、.rela_text、.rela_data。

简述这些节的用途。

答案：

.text	编译汇编后生成的指令机器
.data	已初始化的全局变量、静态局部变量
.rodata	只读数据，如 printf 中的格式串、switch 跳转表等
.bss	未初始化全局变量、静态局部变量、初始化为 0 的全局变量、静态局部变量
.strtab	字符串表，用于符号表的访问（函数名字符串，全局变量名字符串）
.symtab	符号表，记录模块中出现的所有符号的信息（Elf64_Sym 结构），例如：符号出现在第几节、在节内的偏移、所占空间大小、符号的类型、符号名字字符串在字符串表中的偏移，等等。
.shstrtab	节名字符串表
.rela.text	代码节的重定位信息
.rela.data	数据节的重定位信息

4. 编译器在生成可重定位目标文件（.o）时，对于指令代码中需要重定位的符号是如何处理的？假设指令代码中一个符号需要以 R_X86_64_PC32 的方式重定位，链接器是如何处理的？

答案：

(1) 对于指令代码中需要重定位的符号，编译器将它们占用的空间全部填成 00（占位符），并在重定位节 .rela.text 生成该符号的重定位信息。

(2) 链接器从 .rela.text 中获取需要重定位的位置和需要加的常量 A（附加项），然后用下面的值去填充：

S (符号的实际地址) – P (重定位位置的地址) + A (需要加的常量)

= S – (P – A) = 该符号的实际地址 – 下一条指令的地址

5. 编译器将源程序生成可重定位目标文件（.o），链接器将.o 文件生成可执行目标文件。对于下面的程序的 2 条语句，编译器和链接器分别需要完成哪些主要的任务？

```
int global = 35;
void main()
{
    int temp = global;
}
```

答案：

(1) 编译器完成的工作：

对于 int global=35：在数据节.data 存放 35（0x00000023）、在字符串节.strtab 存放字符串“global”、在符号表节.symtab 存放有关符号 global 的信息，包括定义该符号的节号、在相应节（.data）节的地址、数据长度（4 个字节）、属性信息等。

对于 int temp=global：在代码节.text 生成相应的机器指令，在重定位的位置（global 的地址）填充占位符 0x00000000；在代码节的重定位节 .rel.text 记录重定位所需的信息，如重定位的位置（global 占位符的起始地址）、定位方式（地址相对程序计算器 PC 的 32 位偏移）、计算地址时的附加项等。

(2) 链接器完成的工作：

链接器根据重定位节的信息，计算重定位的值，并填充重定位的空间位置。R_X86_64_PC32 重定位方式，填充的值为：S (符号的实际地址) – global 的填充地址 + A (附加项)。

6. 全局偏移量表 GOT 和过程链接表 PLT 的作用是什么？

答案：

GOT 位于数据段的开始。每个表项记录一个本模块引用的外部符号的地址（外部全局变量、外部函数）。

PLT 是.text 节的一部分。每个表项是一小段代码，对应一个全局外部函数。第一次调用时需要将外部函数的地址绑定到 GOT，然后运行 PLT 中相应表项的代码（从 GOT 中取出外部函数的地址并转跳到该地址）。

7. 在程序头表的作用是什么？为什么可重定位目标文件（.o）中可以没有程序头表？为什么可执行目标文件必须有程序头表、动态共享库必须同时有程序头表和节头表？

答案：

程序头表描述可执行文件中的节与虚拟空间中的存储段之间的映射关系。编译时不需要将可执行文件中的节映射到虚拟空间中，所以不需要程序头表（.o 文件中可以没有程序头表）。可执行目标文件需要加载运行，所以需要程序头表（可以没有节头表）。动态共享库既要加载运行、又要在加载时做动态链接（需要将同名的节合并为一个节），所以必须同时有程序头表和

节头表。

8. 链接时的符号解析是什么意思？符号的相关信息保存在哪里？ 链接器可以发现哪些错误？

答案：

符号解析：将模块中引用的符号与某个目标模块中的定义符号建立关联。这样，在重定位时将引用符号的地址重定位为相关联的定义符号的地址。

符号的相关信息：编译器分析出符号的相关信息（Elf_Symbol / Elf64_Sym 结构）并保存在 .symtab 节。

链接器可以发现的错误：没有定义的符号或者不能访问的符号。

9. 在一个模块中（.c 文件），什么样的标识符是符号？什么样的标识符不是符号？符号有哪三种类型（三种类型分别是什么含义）？函数内部定义的非静态变量是什么符号、需要解析吗？什么是强符号？什么是弱符号？定义强、弱符号时有什么规定？

答案：

符号：符号是需要重定位的标识符（变量名和函数名）。包括全局变量名、函数名、静态的局部变量名。非静态的局部变量名不是符号、函数的参数名不是符号。

符号的三种类型：

(1) 模块内部定义的全局符号（其他模块可以引用）。例如，本模块定义的非静态函数和非静态全局变量；

(2) 外部定义的全局符号（其他模块定义并可以被当前模块引用）；

(3) 本模块的局部符号（本模块定义但其他模块不能引用）。例如，本模块定义的 static 的全局变量。

函数内部定义的非静态变量：不是符号、不需要解析。因为它们是在堆栈中分配的临时变量，一般用 [rbp - n] 或 [rsp - n]（n 是常量）的形式去访问。

强符号和弱符号：函数名和已初始化的全局变量名是强符号，未初始化的全局变量名是弱符号。强符号只能被定义一次，弱符号可以定义多次。如果一个符号被不同模块同时定义为强符号和弱符号，则以强定义为准。对弱符号的引用被解析为其强定义符号。

10. 程序的链接主要包括哪些步骤？

答案：

(1) 符号解析

将每个模块中引用的符号与某个目标模块中定义的符号建立关联。

(2) 合并相同的节

将所有目标模块中相同的节合并成新的节。

(3) 对定义的符号进行重定位（确定符号的地址）

确定新节中所有定义的符号在虚拟地址空间中的地址。例如，为变量确定首地址；确定函数的首地址，然后确定函数内每条指令的地址。

(4) 对引用的符号进行重定位

修改.text 节和.data 节中对每个符号的引用（地址）。