武汉大学国家网络安全学院

课程作业报告

题 目： 软件安全实验

专业(班)： 信安5班

学 号： 20222302181161

姓 名： 王亚鹏

课程名称： 软件安全实验

任课教师： 赵磊

2024年 10 月 28日

目录

[实验内容 2](#_Toc181020665)

[实验步骤 2](#_Toc181020666)

[预处理、编译、汇编、链接 2](#_Toc181020667)

[查看并分析目标文件和可执行文件格式 3](#_Toc181020668)

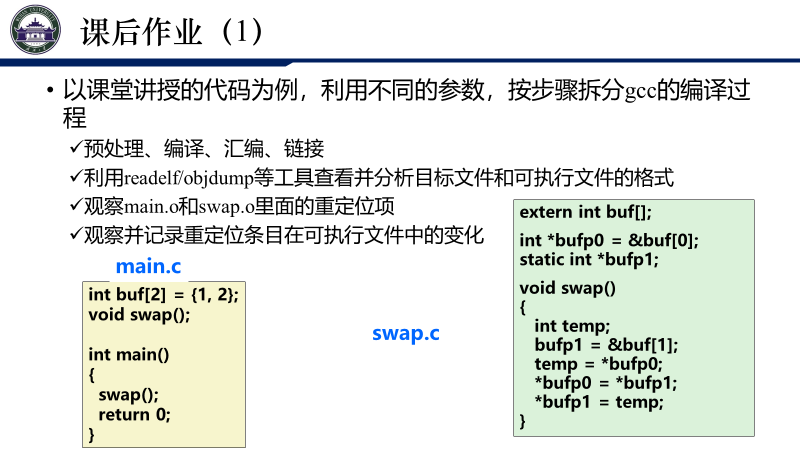
[观察目标文件中的重定位项 3](#_Toc181020669)

[观察并记录重定位条目在可执行文件中的变化 5](#_Toc181020670)

[实验心得 5](#_Toc181020671)

[压缩包文件结构汇总 6](#_Toc181020672)

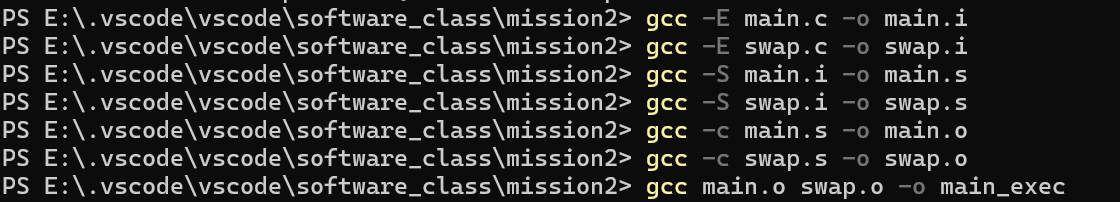
## 实验内容



## 实验步骤

### 预处理、编译、汇编、链接

如图所示的命令进行要求的四个操作



各个命令以及执行命令生成的文件如下：

gcc -E

**解释**：-E 选项表示只执行预处理阶段，将代码中的头文件和宏展开，生成预处理文件

**输出文件**：main.i 和 swap.i，它们包含展开后的代码，方便查看宏定义和头文件展开的效果

gcc -S

**解释**：-S 选项表示将预处理后的代码编译为汇编代码（生成 .s 文件），但不生成目标文件

**输出文件**：main.s 和 swap.s，包含汇编指令，用于更深入理解编译器将C代码转化为汇编的过程

gcc -c

**解释**：-c 选项表示将汇编代码转换为目标文件（.o 文件），不进行链接

**输出文件**：main.o 和 swap.o，包含机器指令，但还没有链接到其他模块

gcc -o

**解释**：不带选项的 gcc 命令会自动执行链接操作，将多个目标文件链接成一个可执行文件

**输出文件**：main\_exec，包含完整的可执行程序，所有符号已解析，且包含机器指令

生成的所有文件如图：



### 查看并分析目标文件和可执行文件格式

使用objdump 分析目标文件和可执行文件格式

查看PE文件的文件头和详细信息——objdump -x main\_exec.exe > details.txt

反汇编PE文件——objdump -d main\_exec.exe > disassembly.txt



这是PE文件头的格式，可以根据这个在details.txt中对应查找

details.txt、disassembly.txt的文件大小都超过了200kb，相比于原文件大了很多，这是因为它们都包括了很多定位信息和导入表、导出表等信息

### 观察目标文件中的重定位项

objdump -r 命令可以显示目标文件中的重定位项信息

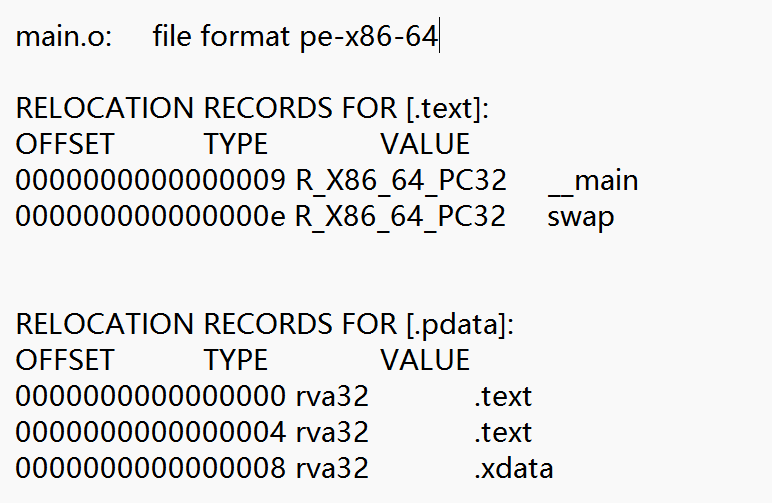
objdump -r main.o > main\_relocations.txt

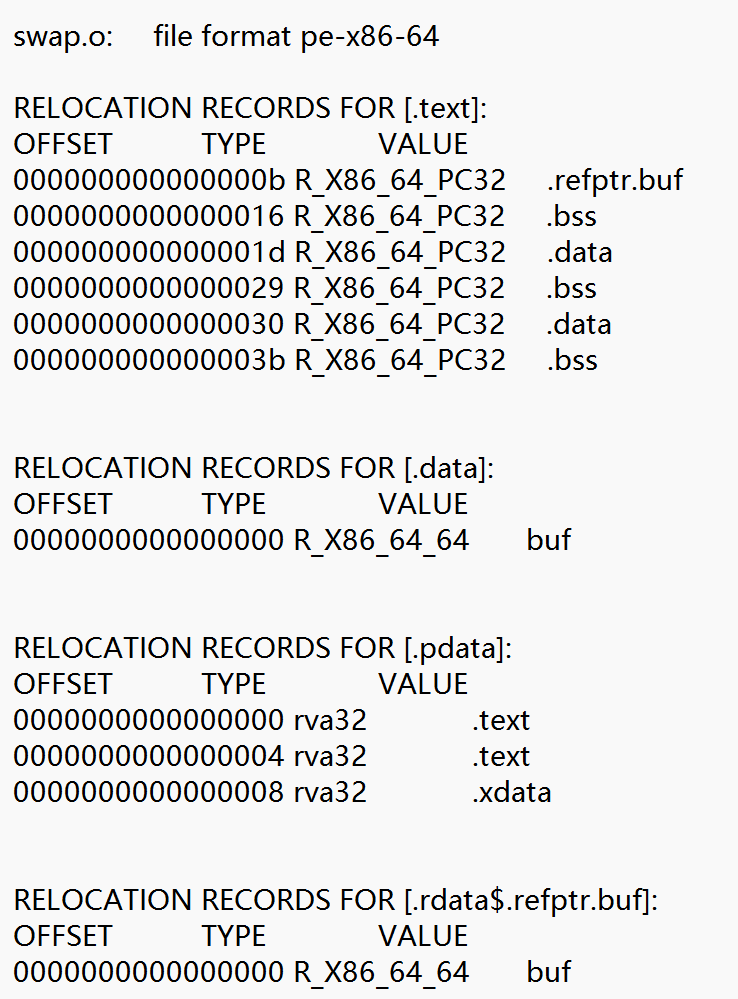
objdump -r swap.o > swap\_relocations.txt

这样我们就得到了两个目标文件中的重定位项信息

 main\_relocations.txt：包含 main.o 文件的重定位项信息

 swap\_relocations.txt：包含 swap.o 文件的重定位项信息





**重定位项内容解读**

重定位表包含需要在链接阶段调整的地址信息，例如：

* **符号名称**：例如外部函数或全局变量名称，如 swap 和 buf
* **重定位偏移**：重定位表中的地址偏移，用于链接器找到需要修正的指令或数据
* **重定位类型**：指明如何修正符号，如 R\_X86\_64\_PC32 表示相对32位地址等

### 观察并记录重定位条目在可执行文件中的变化

在生成的可执行文件中，符号表中的符号地址会被实际分配，可以使用 objdump -t 命令查看符号表：

objdump -t main\_exec > main\_exec\_symbols\_objdump.txt

**符号表对比**：在main\_exec\_symbols.txt和目标文件的重定位项文件中查找相同的符号，如swap和buf

* 比较这些符号在目标文件中的相对地址与在可执行文件中的实际分配地址
* 在目标文件中，这些符号的地址是相对或未解析的，但在可执行文件中，每个符号（函数或变量）都会分配一个确定的地址

## 实验心得

分析了编译生成的目标文件（.o 文件）以及最终的可执行文件，探索了文件结构中的重定位项和符号表信息。通过使用 gcc 编译和 objdump、readelf 等工具，进一步理解了编译和链接过程的细节

**1.动态链接与文件结构的关系**

在生成的可执行文件中，发现反汇编文件较大，这与动态链接有关。动态链接会增加可执行文件中的导入表、跳转表和初始化代码等内容，从而增大了反汇编文件的体积。这让我理解到，动态链接虽然可以减少可执行文件的大小，但在反汇编分析中可能会增加很多与外部符号相关的内容，增加分析难度

**2.重定位项的作用**

在目标文件（如main.o和swap.o）中，重定位项记录了符号的相对地址，指示链接器如何在链接时将这些符号替换为实际的内存地址。通过分析重定位项，深入理解了模块化编程的原理，即在多个文件之间共享符号的同时，确保每个符号在最终的可执行文件中都被正确解析

**3.符号表和调试的关系**

通过 objdump 和 readelf 命令查看符号表，认识到符号表不仅在链接阶段很重要，而且在调试时也不可或缺。符号表包含了所有全局变量和函数的定义信息，帮助调试器在源代码和机器指令之间建立对应关系

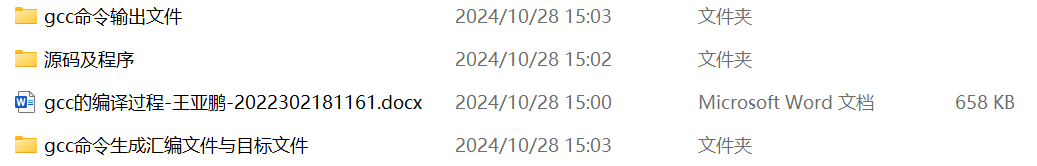
**4.静态链接与动态链接的对比**

本次实验中使用了动态链接的方式生成可执行文件，但也考虑到静态链接的影响。静态链接会将所有依赖的库直接包含在可执行文件中，从而减少了重定位和导入表的内容，生成的反汇编文件更为简洁

**5.工具的使用**

本次实验让我熟悉了 objdump 和 readelf 的基本用法，并理解了如何使用它们分析不同文件中的结构信息。这些工具不仅可以用于分析反汇编，还可以帮助了解文件的符号表、重定位项和依赖关系

## 压缩包文件结构汇总



已经分类打包好，可结合实验报告中提到的文件名对应查找