1.参考复现重定位地址的计算过程

R_386_PC32 的重定位方式

R_386_PC32 重定位方式下，**重定位后的引用地址**应等于：

ADDR(sym) - PC 值

PC = ADDR(下条指令)

= ADDR(sym 引用)-(ADDR(sym 引用) - ADDR(下条指令))

ADDR(sym 引用) = ADDR(引用所在节)+ OFFSET(sym 引用)

所以：

重定位后的引用地址

= ADDR(sym) - ADDR(sym 引用) + 重定位前用地址

Disassembly of section

.text:

08048380<main>:

.....

6:e8 fc ff ff ff call 7<main + 0x7> 7:R_386_PC32 swap

B:b8 00 00 00 00 mov $0x0,%eax

假设 swap 起始虚拟地址 ADDR(sym)=0x8048394

重定位前引用地址 =ADDR(sym 引用)-ADDR(下条指令) = -4 = 0xfcffffff

ADDR(sym 引用) = 0x8048380 + 0x7

PC = 0x08048380 + 0xb

重定位后的引用地址 = 0x8048394 - 0x08048387 - 4

假定：

可执行文件中 main 函数对应机器代码从 0x8048380 开始

swap 紧跟 main 后，其机器代码首地址按四字节边界对齐

则 swap 的起止地址是多少？

0x8048380 + 0x12 = 0x8048392

在 4 字节边界对齐的情况下，是 0x8048394

则重定位后 call 指令的机器代码是什么？

转移目标地址 = PC +偏移地址，PC = 0x8048380+0x07 -init

PC = 0x8048380 +0x07 -(-4) = 0x804838b

重定位值=转移目标地址-PC= 0x8048394 - 0x804838b = 0x9

call 指令的机器代码为 “e8 09 00 00 00”

PC 相对地址方式下，重定位值计算公式为：

PC 相对地址方式下，重定位值计算公式为：

ADDR(r_sym) - ((ADDR(.text) + r_offset) - init)

二、为什么要调用 fork 函数，不用 fork 可不可以，什么时候可以

可执行文件的加载

程序被启动，$ ./P --->调用 fork() --->以构造的 argv 和 envp 为参数调

用 execve()--->execve()调用加载器进行可执行文件加载，并最终转去执行 main

通过调用 execve 系统调用函数来调用加载器
加载器 loader 根据可执行文件的程序(段)头表中的信息，将可执行文件的代码和数据从磁盘“拷贝”到存储器中(实际上不会真正拷贝仅建立一种映像，涉及虚存)
加载后，将 PC(EIP)设定指向 Entry point(符号_start 处)，最终指向 main 函数，以启动程序执行

调用 fork()函数用于创建一个新的进程，这个新进程是调用进程的子进程的复制，在某些情况下，可以避免使用 fork()，如下：

单线程程序：如果程序只是简单的单线程执行，没有需要同时执行的任务，可能不需要调用 fork()。在这种情况下，程序可能只需要直接调用 execve()来加载并执行另一个可执行文件。

不需要创建子进程：有些程序根本不需要创建子进程。这可能是因为程序的设计不需要并发执行，或者因为它们只是简单的脚本。

使用线程代替进程：有些情况下，使用线程可以替代创建子进程。线程是轻量级的执行单元，相比进程更加高效。如果程序需要并发执行

但又不需要完全独立的内存空间，那么可以考虑使用线程。

处理异常情况：在某些情况下，fork()调用可能会失败。程序可能需要考虑并处理 fork()调用失败的情况，这可能会增加程序的复杂性。

2. 加载时的动态链接

为什么 INTERP 调用的动态连接器是.so.2,.2 的后缀是什么含义？

程序头表中有一个特殊的段：INTERP
其中记录了动态链接器目录及文件名 ld-linux.so
在可执行文件的程序头表中，INTERP 段记录了动态链接器的路径和文件名。
动态链接器的文件名通常会有一个版本号作为后缀，比如.so.2,这个版本号表示动态链接器的版本，通常用于支持多个不同版本的动态链接器并存，版本号的变化可能会导致不同的行为，例如支持不同的系统调用或 ABI（应用程序二进制接口）版本。
.so.2 这样的后缀表示动态链接器的版本号为 2，是为了在系统中能够支持多个版本的动态链接器，并能够根据需要选择合适的版本。