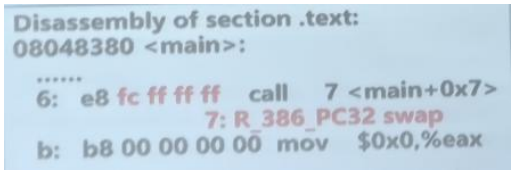
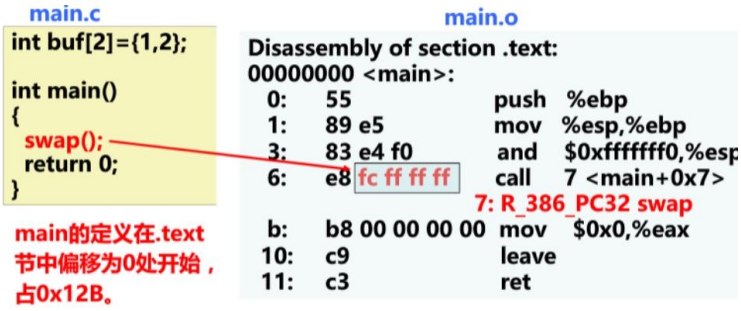
《计算机系统设计》平时作业 4

计算机科学与技术 2110939 李颖

1. 复现重定位地址的计算过程：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 4 字节边界对齐



(1) swap 起始地址为多少？



由图可知，main 的定义占 0x12B，这个地址是在编译时期分配的，表示 main 函数的一部分代码在程序中的位置。而因为 swap 紧随 main 后，因此 swap 的起始地址为 0x8048380+0x12=0x8048392。由于机器代码首地址按 4 字节边界对齐，对齐后的地址为 0x8048394。

(2) 重定位后 call 指令的机器代码是什么？

计算公式：重定位后的偏移地址=转移目标地址-PC

由(1)可知，转移目标地址=0x8048394；由图可知，call 重定位前引用地址为 0xfcffffff，即-4。

而 PC=ADDR(sym 引用)-重定位前引用初始地址，

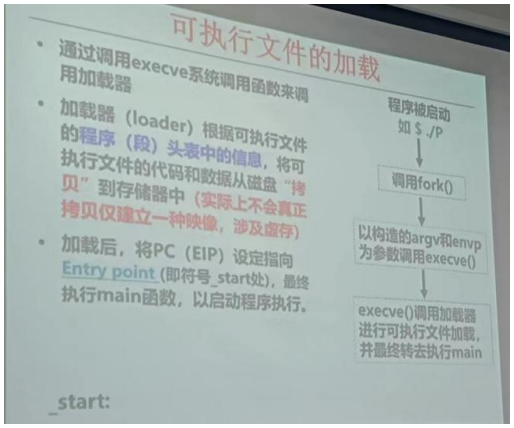
由于起始地址为 0x8048380，call 指令所在地址处标注<mian+0x7>，因此 ADDR(sym 引用)=0x8048380+0x7=0x8048387。

PC=ADDR(sym 引用)-重定位前引用初始地址 =0x8048387 - (-4) =0x804838b

因此，重定位后的偏移地址=转移目标地址-PC =0x8048394 - 0x804838b = 0x9

call 指令的机器代码为 e8 09 00 00 00

2. 为什么要调用 fork 函数，不用 fork 可不可以，什么时候可以？



(1) 为什么要调用 fork 函数？

fork 是 UNIX 系统调用的一个重要部分，用于创建一个新的进程。这个新进程被称为子进程，它是父进程的一个副本。

程序启动后调用 fork()，然后使用 execve()加载新的可执行文件并执行 main 函数。这种流程通常用于创建一个新的进程，然后在新进程中执行不同的程序。调用 fork() 函数的原因如下所示：

【实现进程隔离】：fork()函数创建了一个新的进程，新进程是父进程的副本。这种隔离确保了新进程和父进程之间的资源独立性，避免了新程序对父进程的影响。

【并发执行】：通过调用 fork()，父进程和子进程可以并发执行。

【进程控制】：通过 fork()创建一个新进程，可以在新进程中执行不同的程序，同时保留父进程的执行环境。这种机制允许程序在不同的进程中执行不同的任务，实现更灵活的控制和管理。

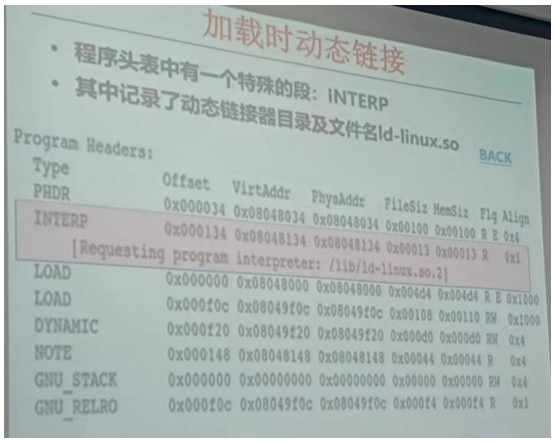
(2) 不用 fork 可不可以，什么时候可以？

不使用 fork 也是可行的，取决于希望如何管理进程以及进程之间的关系。

在某些操作系统中，可能有直接创建新进程的系统调用，例如 Windows 中的 CreateProcess；此外，使用线程可以实现轻量级的并行性和资源共享，但线程间的隔离度不如独立的进程。在上述情景中无需使用 fork。

如果不需要保留原有进程，可以直接使用 exec 系列函数替代 fork 后紧跟 exec。

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



在动态链接的过程中，可执行文件中的 INTERP 段指定了用于加载可执行文件的动态连接器。动态连接器负责在运行时加载共享库以及解析符号等任务。在.so.2 中，.so 表示这是一个共享对象文件，而后面的版本号.2 则表示共享对象的版本号。在这种情况下，.2 表示动态连接器的版本是 2。版本号的变化通常反映了动态连接器的不同版本之间的差异，可能包括功能改进、bug 修复或性能优化等。

在 Linux 系统中，共享库的命名通常遵循一定的规范，其中包括版本号。.so 是共享库文件的通用后缀，而.so.2 则表示特定版本的共享库。这种命名约定的设计是为了解决共享库的版本管理和兼容性问题。

使用.so.2 这样的命名约定可以明确标识共享库的版本。通过版本号的指定，可以在系统中同时存在多个版本的共享库，并确保程序能够链接到正确的版本。此外，在 Linux 系统中，共享库的版本可能会随着时间的推移而更新和演变。通过使用版本号，可以确保新版本的共享库不会覆盖旧版本，从而保持对旧版本的兼容性。在符号链接时，可以将.so.2 连接到实际的共享文件，使程序在编译时和运行时能够正确找到所需的共享库文件，同时保持版本控制和管理的灵活性。