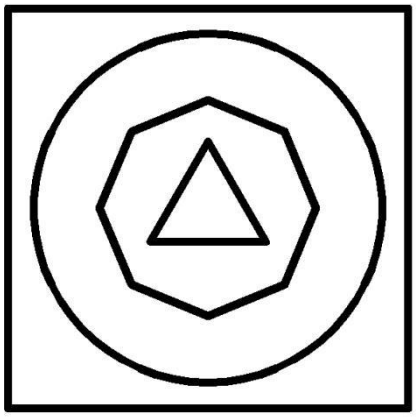
什么是重定位?



本文件由皇天惊虞制作，免费流通于网络。

制作时间　**2023.12.29**

什么是重定位?

一、必须知道的几个概念。

1、链接地址和运行地址。

①运行地址，顾名思义就是程序运行的时候的地址，也就是你用工具将代码下载到 RAM 的那个地址，也叫加载地址。

②链接地址，由链接脚本指定的地址。为什么需要链接脚本指定地址呢？你想一下，在 c 语言编程中，当我们需要调用一个 A 函数的时候，编译器是怎么找到这个 A 函数？编译器肯定是知道它被放在哪里才可以找到它。那就是链接脚本的作用，链接脚本其实在程序被执行之前都已经指定 A 函数一个地址编号，以后所有的函数调用我们都会去这个地址编号那里寻找 A 函数。有点类似于 c 语言的指针变量。

2、位置有关码与位置无关码。

①位置有关码，就是这句代码的执行正确与否还需要取决于当前的地址，也就是说跟地址已经绑定了的，例如：ldr PC, _main，就是 PC 指针必须跳转到_main（函数名就是一个地址）这个地址去，代码执行成功与否就相当于受到了这个地址的约束，假如这个地址的内容不存放_main 这个函数，就会出错了。

②位置无关码，就是这句代码在哪里运行都可以的，跟所处的地址无关，跟位置有关码相反。

二、重定位需要理解的一些问题。

1、链接地址跟运行地址不同的情况下会出现什么情况？

答：以上面举的函数 A 为例，当链接地址跟运行地址不同的时候，假如链接地址是 0x1000，运行地址（加载地址）是 0x0000，链接脚本指定函数 A 将来是要存放到（基地址+偏移量）=0x1000+0x0001=0x1001 地址的，但是程序在下载的时候却把这个程序下载到 0x0000，所以函数 A 的地址实际上是存放在（基地址+偏移量）=0x0000+0x0001=0x0001 这个地址的。当程序运行到一行位置有关码例如：ldr PC, A ，编译器首先就会按照链接脚本指定的 A 的那个地址 0x1001 寻找 A 函数，但是因为加载地址跟链接地址不同的原因，实际上 A 函数已经被放到了 0x0001，所以执行就会出错。所以，当这两个地址不同的时候，执行一段位置有关码的时候就会发生不可预估的错误。

2、为什么会出现链接地址跟运行地址不同的情况？

什么是重定位?

答：当一块芯片启动的时候，依靠内部的 SRAM，可以运行一小段代码，而因为 DDR 还没初始化，注定了开始的运行地址是在内部 SRAM 中的。当我们需要运行一个操作系统，那么点的内存怎么够运行呢？所以这时候就需要初始化 DDR 才可，而因为我们知道这代码将来都是在 DDR 上面运行的，所以链接脚本指定的链接地址肯定是 DDR 上面的地址，所以这就出现了链接地址跟运行地址不同的情况了。

3、什么是重定位？

答：由于出现 1 这样的问题，就需要使用重定位这种方式解决上面的问题了。那什么是重定位呢？重定位就是在链接地址跟运行地址不同的情况下，执行一段位置无关码，这段位置无关码的作用就是将原来的那份代码全部复制到链接地址那里去，然后自己再长跳转到新的那份代码的刚刚执行的那个位置。这样就实现了链接地址跟运行地址一致的情况了。

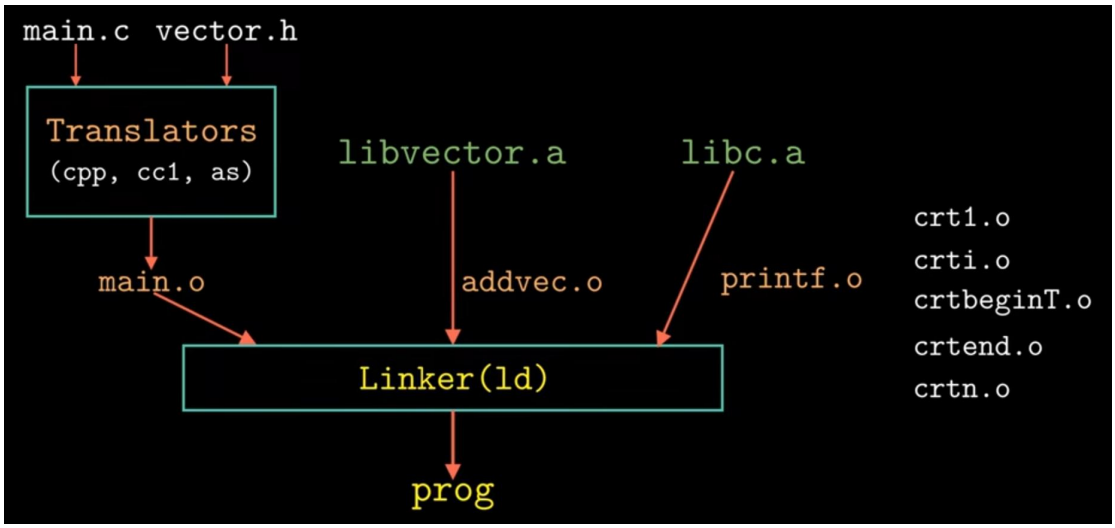
3、为什么需要重定位？

答：就是链接地址跟运行地址不同，在这个情况下我们可以有两种方案：

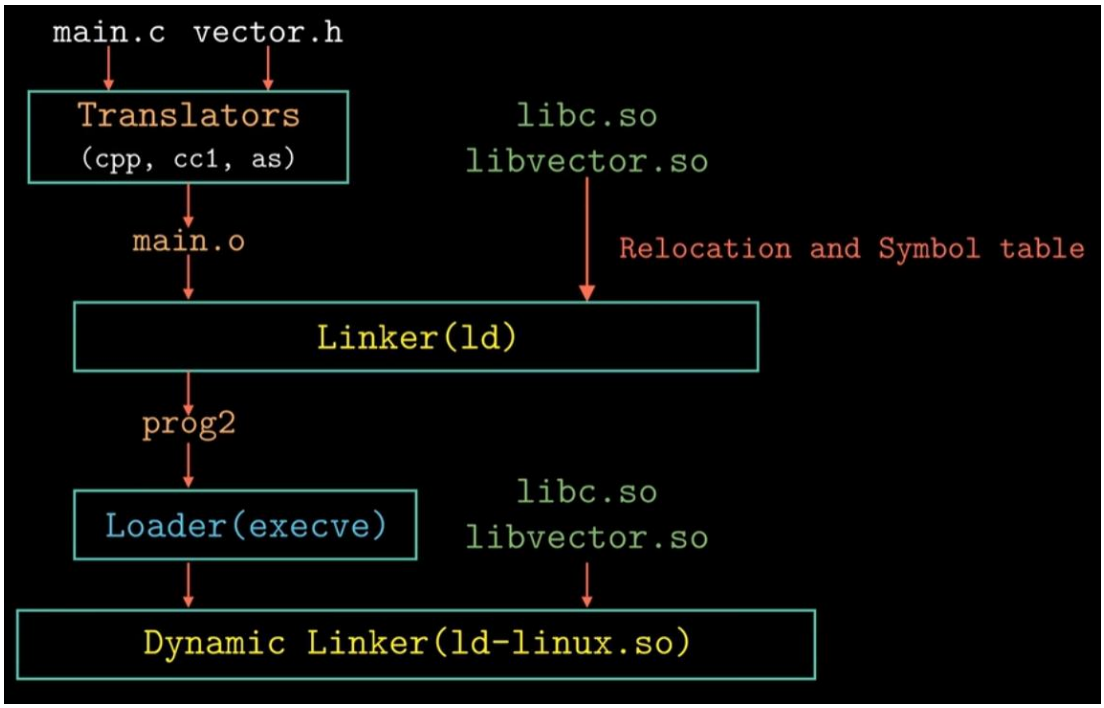
①全部使用位置无关码。

②进行重定位让这两个地址相同。

我们知道，如果是一个小代码，使用①时可以的，但是一个大的代码文件很难保证全部都使用位置无关码的，这也是不现实的，所以必须使用重定位解决这个问题



什么是重定位?



一、定义：重定位就是把程序的 逻辑地址空间变换成内存中的实际 物理地址空间的过程，也就是说在装入时对 目标程序中指令和数据的修改过程。他是实现 多道程序在内存中同时运行的基础。重定位有两种，分别是 动态重定位与 静态重定位

二、分类　　1、静态重定位：即在程序装入内存的过程中完成，是指在程序开始运行前，程序中的各个地址有关的项均已完成重定位，地址变换通常是在装入时一次完成的，以后不再改变，故成为静态重定位。　　2、动态重定位：它不是在程序装入内存时完成的，而是 CPU 每次访问内存时 由 动态地址变换机构（硬件）自动进行把相对地址转换为 绝对地址。动态重定位需要软件和硬件相互配合完成。

三、时机　　将逻辑地址空间重定位到物理地址空间的时机有三种：

1、 程序编译连接时。

2、程序装入内存时。

3、程序执行时。

四、扩展阅读：

地址重定位就是操作系统将逻辑地址转变为物理地址的过程。。。也就是对目标程序中的指令和数据进行修改的过程

将逻辑地址空间重定位到物理地址空间的时机有三种：

1、程序编译连接时。

2、程序装入内存时。

3、程序执行时。

在这之前我一直对地址重定位的细节不是很了解。以下文章摘自《操作系统原理 DOS 篇（第二版）》。是我目前看过的关于重定位的最好的文章。与大家分享 一下。

以下介绍程序是如何装入内存，从而变成在计算机内可执行的形式的。

什么是重定位?

在用汇编语言或高级语言编写的程序中，是通过符号名来访问子程序和数据的，我们把程序中符号名的集合叫做“名字空间”。汇编语言源程序经过汇编，或者高级语言源程序经过编译，得到的目标程序是以“0”作为参考地址的模块，然后多个目标模块由连接程序连接成一个具有统一地址的装配模块，以便最后装入内存中执行。我们把 **目标模块 obj 中的地址称为相对地址（或逻辑地址）**，而把相对地址的集合叫做“相对地址空间”或简单地叫做“地址空间”。

装配模块虽然具有统一的地址空间，但它仍是以“0”作为参考地址，即是浮动的。要把它装入内存执行，就要确定装入内存的实际物理地址，并修改程序中与 地址有关的代码，这一过程叫做地址重定位。

地址空间的程序和数据经过地址重定位处理后，就变成了可由 CPU 直接执行的绝对地址程序。我们把这一地址集合称为“绝对地址空间”或“存储空间”。

地址重定位完成的相对地址转换成内存的绝对地址工作又称为地址映射（map）、按照重定位的时机，可分为静态重定位和动态重定位。

一、静态重定位

静态重定位是在程序执行之前进行重定位，它根据装配模块将要装入的内存起始位置，直接修改装配模块中的有关使用地址的指令。

例如，一个以“0”作为参考地址的装配模块，要装入以 100 为起始地址的存储空间。显然，在装入之前要做某些修改，程序才能正确执行。例如，MOV EAX，[500]这条指令的意义，是把相对地址为 500 的存储单元内容 1234 装入 EAX 号累器。现在内容为 1234 的存储单元的实际地址为 1500， 即为相对地址(500)加上装入的地址(1000),因此，MOV EAX,[500]这条指令中的直接地址码也要相应地加上起始地址,而成为 MOV EAX,[1500]。

程序中涉及直接地址的每条指令都要进行这样的修改。需要修改的位置称为重定位项，所做的加实际装入模块起始地址修改中的块起始地址称为重定位因子。

为支持静态重定位，连接程序在生成统一地址空间和装配模块时， 应产生一个重定位项表，连接程序此时还不知道装配模块将要装入的实际位置，故重定位表 所给出的需修改位置是相对地址所表示的位置。

操作系统的装入程序要把装配模块和重定位项表一起装入内存。由装配模块的实际装入起始地址得到重定位因子，然后实施如下两步：

（1）取重定位项，加上重定位因子而得到欲修改位置的实际地址；

（2）对实际地址中的内容再做加重定位因子的修改，从而完成指令代码的修改。

对所有的重定位项实施上述两步操作后，静态重定位才完成，尔后可启动程序执行。使用过的重定位项表内存副本随即被废弃。

静态重定位有着无需硬件支持的优点，但存在着如下的缺点：一是程序重定位之后就不能在内存中搬动了；二是要求程序的存储空间是连续的，不能把程序放在若干个不连续的区域内。

二、动态重定位

动态重定位是指，不是在程序执行之前而是在程序执行过程中进行地址重定位。更确切地说，是在 CPU 每次访问内存单元前才进行地址变换。动态重定位可使装配模 块不加任何修改而装入内存，但是它需要硬件——定位寄存器的支持。

程序的目标模块装入内存时，与地址有关的各项均保持原来的相对地址不进行任何修改。如 MOV 1，[500]这条指令仍是相对地址 500。当此模块被 操作系统调度到处理机上执行

什么是重定位?

时，操作系统将把此模块装入的实际起始起始地址减去目标模块的相对基地址，然后将其差值装入定位寄存器中。当 CPU 取得一条访 问内存的指令时，地址变换硬件逻辑自动将指令中的相对地址与定位寄存器中的值相加，再依此和值作为内存绝对地址去访问该单元中的数据。

由此可见，进行动态重定位的时机是在指令执行过程中，每次访问内存前动态地进行。采取动态重定位可带来两个好处：

（1）目标模块装入内存时无需任何修改，因而装入之后再搬迁也不会影响其正确执行，这对于存储器紧缩、解决碎片问题是极其有利的；

（2）一个程序由若干个相对独立的目标模块组成时，每个目标模块各装入一个存储区域，这些存储区域可以不是顺序相邻的，只要各个模块有自己对应的定位寄存器就行。

动态重定位技术所付出的代价是需要硬件支持。

总结：

静态地址重定位：即在程序装入内存的过程中完成，是指在程序开始运行前，程序中的各个地址有关的项均已完成重定位，地址变换通常是在装入时一次完成的，以后不再改变，故成为静态重定位。

优点：无需硬件支持

缺点：1）程序重定位之后就不能在内存中搬动了；2）要求程序的存储空间是连续的，不能把程序放在若干个不连续的区域中。

动态地址重定位：不是在程序执行之前而是在程序执行过程中进行地址重定位。更确切的说，是在每次访问内存单元前才进行地址变换。动态重定位可使装配模块不加任何修改而装入内存，但是它需要硬件一定位寄存器的支持。

优点：1）目标模块装入内存时无需任何修改，因而装入之后再搬迁也不会影响其正确执行，这对于存储器紧缩、解决碎片问题是极其有利的；2）一个程序由若干个相对独立的目标模块组成时，每个目标模块各装入一个存储区域，这些存储区域可以不是顺序相邻的，只要各个模块有自己对应的定位寄存器就行。

缺点：需要硬件支持。