



(12) 发明专利申请

(10) 申请公布号 CN 104951290 A

(43) 申请公布日 2015.09.30

(21) 申请号 201410126197.3

(22) 申请日 2014.03.31

(71) 申请人 国际商业机器公司

地址 美国纽约阿芒克

(72) 发明人 郭久福 张嗣元 纪金松 班怀芸

(74) 专利代理机构 北京市柳沈律师事务所

11105

代理人 于小宁

(51) Int. Cl.

G06F 9/44(2006.01)

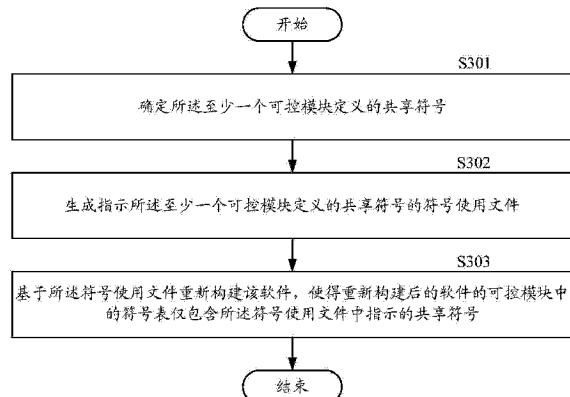
权利要求书2页 说明书11页 附图3页

(54) 发明名称

优化软件的方法和设备

(57) 摘要

提供了一种优化软件的方法和设备。所述软件包括多个模块，所述多个模块包括至少一个可控模块，所述方法包括：确定所述至少一个可控模块定义的共享符号，所述共享符号是被所述软件的两个或更多模块使用的符号；生成指示所述至少一个可控模块定义的共享符号的符号使用文件；以及基于所述符号使用文件重新构建该软件，使得重新构建后的软件的可控模块中的符号表仅包含所述符号使用文件中指示的共享符号。利用所述方法和设备，可以准确有效地减少符号表中的符号数量，从而减少加载该符号表所需的时间，提高软件的运行效率。



1. 一种优化软件的方法,所述软件包括多个模块,所述多个模块包括至少一个可控模块,所述方法包括:

确定所述至少一个可控模块定义的共享符号,所述共享符号是被所述软件的两个或更多模块使用的符号;

生成指示所述至少一个可控模块定义的共享符号的符号使用文件;以及

基于所述符号使用文件重新构建该软件,使得重新构建后的软件的可控模块中的符号表仅包含所述符号使用文件中指示的共享符号。

2. 如权利要求1所述的方法,其中,所述确定所述至少一个可控模块定义的共享符号包括:

对于所述至少一个可控模块中的每个可控模块,分析该可控模块定义的符号是否被该可控模块和该可控模块调用的模块中的两个或更多模块使用,以确定该可控模块定义的符号是否是共享符号。

3. 如权利要求2所述的方法,其中,通过对所述每个可控模块和所述每个可控模块调用的模块执行过程间分析,来分析所述至少一个可控模块中的每个可控模块定义的符号是否被所述每个可控模块和所述每个可控模块调用的模块中的两个或更多模块使用。

4. 如权利要求1至3之一所述的方法,其中,所述至少一个可控模块包括由该软件的开发者开发的可执行文件和共享库。

5. 如权利要求1至3之一所述的方法,其中,所述符号使用文件包含指示定义所述共享符号的可控模块的信息以及指示使用所述共享符号的模块的信息。

6. 如权利要求1所述的方法,其中,所述基于所述符号使用文件重新构建该软件,使得重新构建后的软件的可控模块中的符号表仅包含所述符号使用文件中指示的共享符号包括:

对所述至少一个可控模块中的每个可控模块的源文件重新进行编译以产生目标文件,其中在所述目标文件中,对所述每个可控模块中除了所述符号使用文件中指示的共享符号以外的符号设置比所述共享符号更快的访问方式;以及

对所述每个可控模块的目标文件重新进行链接以产生重新构建后的软件中与该可控模块对应的可控模块,其中所述对应的可控模块中的符号表仅包含所述符号使用文件中指示的共享符号。

7. 一种优化软件的设备,所述软件包括多个模块,所述多个模块包括至少一个可控模块,所述设备包括:

确定装置,被配置为确定所述至少一个可控模块定义的共享符号,所述共享符号是被所述软件的两个或更多模块使用的符号;

生成装置,被配置为生成指示所述至少一个可控模块定义的共享符号的符号使用文件;以及

构建装置,被配置为基于所述符号使用文件重新构建该软件,使得重新构建后的软件的可控模块中的符号表仅包含所述符号使用文件中指示的共享符号。

8. 如权利要求7所述的设备,其中,所述确定装置通过对所述至少一个可控模块中的每个可控模块分析该可控模块定义的符号是否被该可控模块和该可控模块调用的模块中的两个或更多模块使用,以确定该可控模块定义的符号是否是共享符号,来确定所述至少

一个可控模块定义的共享符号。

9. 如权利要求 8 所述的设备,其中,所述确定装置通过对所述每个可控模块和所述每个可控模块调用的模块执行过程间分析,来分析所述至少一个可控模块中的每个可控模块定义的符号是否被所述每个可控模块和所述每个可控模块调用的模块中的两个或更多模块使用。

10. 如权利要求 7 至 9 之一所述的设备,其中,所述至少一个可控模块包括由该软件的开发者开发的可执行文件和共享库。

11. 如权利要求 7 至 9 之一所述的设备,其中,所述符号使用文件包含指示定义所述共享符号的可控模块的信息以及指示使用所述共享符号的模块的信息。

12. 如权利要求 7 所述的设备,其中,所述构建装置包括:

编译器,被配置为对所述至少一个可控模块中的每个可控模块的源文件重新进行编译以产生目标文件,其中在所述目标文件中,对所述每个可控模块中除了所述符号使用文件中指示的共享符号以外的符号设置比所述共享符号更快的访问方式;以及

链接器,被配置为对所述每个可控模块的目标文件重新进行链接以产生重新构建后的软件中与该可控模块对应的可控模块,其中所述对应的可控模块中的符号表仅包含所述符号使用文件中指示的共享符号。

优化软件的方法和设备

技术领域

[0001] 本发明涉及软件的优化，并且具体涉及一种优化软件的方法和设备。

背景技术

[0002] 软件可以包括若干模块，例如可执行文件以及该可执行文件需要调用的库。每个模块可以包含大量符号，例如函数和 / 或变量等，这些符号可以包括由该模块自己定义并且仅由该模块使用的符号（在下文中，可称为内部符号）、由该模块自己定义并且由两个或更多模块使用的符号以及由其他模块定义并且由两个或更多模块使用的符号（在下文中，可以将后两种符号称为共享符号）。通常，在每个模块中建立符号表以存储该模块中的所有符号以及与该符号相关的信息，例如该符号的类型、用于定位该符号的、该符号的绝对地址和偏移量等。当运行该模块时，可以加载所述符号表，并且根据符号表中的符号以及相关信息来执行对应的操作。

[0003] 随着软件变得越来越大，符号表中的符号以及相关信息也越来越多，使得符号表越来越大，这导致在软件运行时加载和解析符号表耗费的时间增多，从而降低了软件的运行效率。传统上，可以使用两种方法来解决这一问题。在第一种传统方法中，软件的开发者人工地分析软件的各个模块的源文件以找出软件中的所有共享符号，并且编写导出文件，在该导出文件中将所述共享符号标记为“导出”，然后将该导出文件提供给链接器，链接器根据该导出文件来在模块的符号表中隐藏内部符号。然而，由于软件的源文件往往非常大，因此人工找出所有共享符号的操作非常困难和耗时，并且容易遗漏，而且，在例如基于 C++ 语言的软件中，符号的名称等在编译和链接期间会发生变化，使得这种方法难以实现。在第二种传统方法中，软件的开发者修改每个模块的源文件，以便给该模块中的符号增加指示该符号是否应该被隐藏的可见性标签，然后链接器根据符号的可见性标签来确定是否在符号表中隐藏该符号。这种方法需要修改源文件，因而会引入额外的风险，而且，当更新软件时，需要相应地更新每个符号的标签，因而比较麻烦。

发明内容

[0004] 为了解决上述问题，本发明的一个目的是提供一种优化软件的方法和设备，其能够准确有效地减少符号表的大小，从而减少加载该符号表所需的时间，提高软件的运行效率。

[0005] 根据本发明的一个方面，提供了一种优化软件的方法，所述软件包括多个模块，所述多个模块包括至少一个可控模块，所述方法包括：确定所述至少一个可控模块定义的共享符号，所述共享符号是被所述软件的两个或更多模块使用的符号；生成指示所述至少一个可控模块定义的共享符号的符号使用文件；以及基于所述符号使用文件重新构建该软件，使得重新构建后的软件的可控模块中的符号表仅包含所述符号使用文件中指示的共享符号。

[0006] 根据本发明的另一方面，提供了一种优化软件的设备，所述软件包括多个模块，所

述多个模块包括至少一个可控模块，所述设备包括：确定装置，被配置为确定所述至少一个可控模块定义的共享符号，所述共享符号是被所述软件的两个或更多模块使用的符号；生成装置，被配置为生成指示所述至少一个可控模块定义的共享符号的符号使用文件；以及构建装置，被配置为基于所述符号使用文件重新构建该软件，使得重新构建后的软件的可控模块中的符号表仅包含所述符号使用文件中指示的共享符号。

[0007] 利用根据本发明上述方面的方法和设备，可以自动地分析软件的可控模块中的各个符号的使用状态，并且基于分析结果来重新构建该软件，使得重新构建后的软件的可控模块中的符号表仅包含被该软件的多个模块共享的共享符号。这样，可以准确且有效地减少符号表中的符号数量，缩短软件运行时加载和解析该符号表所耗费的时间，提高软件的运行效率。

附图说明

[0008] 通过结合附图对本公开示例性实施方式进行更详细的描述，本公开的上述以及其他目的、特征和优势将变得更加明显，其中，在本公开示例性实施方式中，相同的参考标号通常代表相同部件。

[0009] 图 1 示出了适于用来实现本发明实施方式的示例性计算机系统 / 服务器 12 的框图。

[0010] 图 2 示意性地示出了包括多个模块的软件的架构。

[0011] 图 3 示出了根据本发明实施例的优化软件的方法的流程图。

[0012] 图 4 示意性地示出了对一个可控模块生成模块符号使用文件的过程。

[0013] 图 5 示出了根据本发明实施例的优化软件的设备的框图。

[0014] 图 6 示出了图 5 所示的构建装置的结构框图。

具体实施方式

[0015] 下面将参照附图更详细地描述本公开的优选实施方式。虽然附图中显示了本公开的优选实施方式，然而应该理解，可以以各种形式实现本公开而不应被这里阐述的实施方式所限制。相反，提供这些实施方式是为了使本公开更加透彻和完整，并且能够将本公开的范围完整地传达给本领域的技术人员。

[0016] 图 1 示出了适于用来实现本发明实施方式的示例性计算机系统 / 服务器 12 的框图。图 1 显示的计算机系统 / 服务器 12 仅仅是一个示例，不应本发明实施例的功能和使用范围带来任何限制。

[0017] 如图 1 所示，计算机系统 / 服务器 12 以通用计算设备的形式表现。计算机系统 / 服务器 12 的组件可以包括但不限于：一个或者多个处理器或者处理单元 16，系统存储器 28，连接不同系统组件(包括系统存储器 28 和处理单元 16)的总线 18。

[0018] 总线 18 表示几类总线结构中的一种或多种，包括存储器总线或者存储器控制器，外围总线，图形加速端口，处理器或者使用多种总线结构中的任意总线结构的局域总线。举例来说，这些体系结构包括但不限于工业标准体系结构 (ISA) 总线，微通道体系结构 (MAC) 总线，增强型 ISA 总线、视频电子标准协会 (VESA) 局域总线以及外围组件互连 (PCI) 总线。

[0019] 计算机系统 / 服务器 12 典型地包括多种计算机系统可读介质。这些介质可以是

任何能够被计算机系统 / 服务器 12 访问的可用介质,包括易失性和非易失性介质,可移动的和不可移动的介质。

[0020] 系统存储器 28 可以包括易失性存储器形式的计算机系统可读介质,例如随机存取存储器(RAM)30 和 / 或高速缓存存储器 32。计算机系统 / 服务器 12 可以进一步包括其它可移动 / 不可移动的、易失性 / 非易失性计算机系统存储介质。仅作为举例,存储系统 34 可以用于读写不可移动的、非易失性磁介质(图 1 未显示,通常称为“硬盘驱动器”)。尽管图 1 中未示出,可以提供用于对可移动非易失性磁盘(例如“软盘”)读写的磁盘驱动器,以及对可移动非易失性光盘(例如 CD-ROM, DVD-ROM 或者其它光介质)读写的光盘驱动器。在这些情况下,每个驱动器可以通过一个或者多个数据介质接口与总线 18 相连。存储器 28 可以包括至少一个程序产品,该程序产品具有一组(例如至少一个)程序模块,这些程序模块被配置以执行本发明各实施例的功能。

[0021] 具有一组(至少一个)程序模块 42 的程序 / 实用工具 40,可以存储在例如存储器 28 中,这样的程序模块 42 包括——但不限于——操作系统、一个或者多个软件、其它程序模块以及程序数据,这些示例中的每一个或某种组合中可能包括网络环境的实现。程序模块 42 通常执行本发明所描述的实施例中的功能和 / 或方法。

[0022] 计算机系统 / 服务器 12 也可以与一个或多个外部设备 14(例如键盘、指向设备、显示器 24 等)通信,还可与一个或者多个使得用户能与该计算机系统 / 服务器 12 交互的设备通信,和 / 或与使得该计算机系统 / 服务器 12 能与一个或多个其它计算设备进行通信的任何设备(例如网卡,调制解调器等等)通信。这种通信可以通过输入 / 输出(I/O)接口 22 进行。并且,计算机系统 / 服务器 12 还可以通过网络适配器 20 与一个或者多个网络(例如局域网(LAN),广域网(WAN)和 / 或公共网络,例如因特网)通信。如图所示,网络适配器 20 通过总线 18 与计算机系统 / 服务器 12 的其它模块通信。应当明白,尽管图中未示出,可以结合计算机系统 / 服务器 12 使用其它硬件和 / 或软件模块,包括但不限于:微代码、设备驱动器、冗余处理单元、外部磁盘驱动阵列、RAID 系统、磁带驱动器以及数据备份存储系统等。

[0023] 下面,将参照附图描述根据本发明实施例的优化软件的方法和设备。

[0024] 当编写软件时,开发者首先编写源文件(源代码),然后利用编译器来编译该源文件,从而生成目标文件。接下来,利用链接器将目标文件以及需要调用的库链接到一起,从而生成可执行文件或共享库。所述需要调用的库可以包括开发者按照上述方式预先开发的共享库、由第三方开发的库和 / 或系统库。所述可执行文件和库可以称为模块,其中,由于可执行文件和共享库是开发者自己开发的,换言之,开发者可以通过修改源文件来控制可执行文件和共享库,因此,可以将所述可执行文件和共享库(以及开发者自己开发的其他模块)称为可控模块,反之,由于开发者通常不能修改第三方库和系统库,因此可以将第三方库和系统库称为不可控模块。当然,在其他实施例中,也可以将虽然不是开发者自己开发、但是开发者能够获取并修改其源代码的第三方库和 / 或系统库称为可控模块。图 2 示意性地示出了软件的架构。如图 2 中的箭头所示,在软件运行期间,作为可控模块的可执行文件和共享库以及作为不可控模块的第三方库和系统库可以互相调用,从而执行相应的操作。

[0025] 如上文所述,软件的每个模块可以定义多个符号,这些符号中的一部分仅在该模块内部使用(即,内部符号),另一部分符号可被该模块以及其他模块使用(即,共享符号)。在本发明的实施例中,自动地分析软件的各个可控模块定义的符号以确定共享符号,并且

基于所确定的共享符号重新构建所述软件,使得所述软件的各个可控模块中的符号表仅包含所述共享符号,从而减小符号表的大小,提高软件运行效率。

[0026] 下面,将参照图 3 来更详细地描述根据本发明实施例的优化软件的方法。该软件可以包括多个模块,所述多个模块包括至少一个可控模块。

[0027] 如图 3 所示,在步骤 S301 中,确定所述至少一个可控模块定义的共享符号。如上文所述,所述共享符号是被所述软件的两个或更多模块使用的符号。

[0028] 在本发明的实施例中,可以自动地分析该软件的各个模块,以找出所述至少一个可控模块定义的所有共享符号。

[0029] 具体地,可以分析所述至少一个可控模块中的每个可控模块定义的符号的使用状态,即,分析所述至少一个可控模块中的每个可控模块定义的符号是否被所述每个可控模块和所述每个可控模块调用的模块(其可以包括可控模块和 / 或不可控模块)中的两个或更多模块使用,以确定所述每个可控模块定义的符号是否是共享符号。例如,通过对所述至少一个可控模块中的每个可控模块以及所述每个可控模块调用的其他模块执行过程间分析(IPA),来确定所述每个可控模块定义的符号的使用状态。例如,对于作为可控模块的可执行文件定义的每个符号,可以通过 IPA 来确定该符号是否被该可执行文件以及该可执行文件调用的库中的两个或更多模块使用,从而确定该符号是否是共享符号。执行 IPA 的方法是本领域公知的,在这里不再赘述。

[0030] 在步骤 S302 中,生成指示所述至少一个可控模块定义的共享符号的符号使用文件。除了指示所述至少一个可控模块定义的共享符号以外,所述符号使用文件还可以包含指示定义所述共享符号的可控模块的信息以及指示使用所述共享符号的模块的信息。

[0031] 具体地,可以对于所述至少一个可控模块中的每个可控模块,基于步骤 S301 中的确定结果,生成指示所述每个可控模块和所述每个可控模块调度调用的其他可控模块定义的共享符号的符号使用文件(以下可称为模块符号使用文件)。然后,可以合并对于所述至少一个可控模块中的全部可控模块生成的模块符号使用文件,以生成指示所述至少一个可控模块定义的所有共享符号的符号使用文件(以下可称为总符号使用文件)。所述模块符号使用文件和总符号使用文件可以采用任何适当的格式,例如表格、文本文件等。具体地,可以通过将各个模块符号使用文件中与相同的共享符号相关的信息(例如指示使用该共享符号的模块的信息)合并,来合并各个模块符号使用文件。例如,假设软件包含两个可控模块,并且分别对这两个可控模块生成下表 1 和 2 所示的模块符号使用文件:

[0032]

符号	Def	Ref
sym1	mod1	mod1, mod2
sym2	mod2	mod1
sym3	mod2	mod2

[0033] 表 1

[0034]

符号	Def	Ref
sym1	mod1	mod1
sym2	mod2	mod2, mod1
sym3	mod2	mod2

[0035] 表 2

[0036] 作为示例,每个模块使用文件可以包括字段“符号”、“Def”和“Ref”三个字段,其中“符号”字段指示可控模块以及可控模块调用的其他可控模块定义的符号,“Def”字段指示定义所述符号的可控模块,“Ref”字段表示使用所述符号的模块(可包括可控模块和不可控模块),sym1、sym2 和 sym3 表示符号,mod1 和 mod2 表示可控模块。通过合并这两个模块符号使用文件,具体地,通过将与相同的符号对应的“Ref”字段合并,可以获得下表 3 所示的总符号使用文件:

[0037]

符号	Def	Ref
sym1	mod1	mod1, mod2
sym2	mod2	mod2, mod1
sym3	mod2	mod2

[0038] 表 3

[0039] 在表 3 所示的总符号使用文件中,通过“Ref”字段,可以识别每个符号是否是共享符号。

[0040] 下面结合图 4 来示例性地描述步骤 S301 和 S302 的具体操作。在图 4 的例子中示出了 4 个可控模块,即可执行文件 exe1 (未示出) 以及开发者自己开发的共享库 lib1 和 lib2 以及第三方库 lib3,其中可执行文件 exe1 是使用两个目标文件(即目标文件 obj1 和目标文件 obj2)生成的,并且该可执行文件 exe1 调用所述共享库 lib1 和 lib2 以及第三方库 lib3。根据上文所述可知,可执行文件 exe1(目标文件 obj1 和目标文件 obj2)以及共享库 lib1 和 lib2 为可控模块,第三方库 lib3 为不可控模块。此外,假设可执行文件 exe1 对应的目标文件 obj1 定义了两个函数 fun1 和 s_fun1 以及一个变量 var1,可执行文件 exe1 对应的目标文件 obj2 定义了两个函数 fun2 和 s_fun2 以及一个变量 var2,共享库 lib1 定义了函数 l_fun1 和 l_s_fun1 以及一个变量 l_var1,共享库 lib2 定义了函数 l_fun2 和 l_s_fun2 以及一个变量 l_var2,第三方库 lib3 定义了函数 l_fun3 和 l_s_fun3 以及一个变量 l_var3。

[0041] 在步骤 S301 中,可以按照上文所述的方式,例如通过 IPA 来分析各个可控模块(可执行文件 exe1 和共享库 lib1 和 lib2) 定义的各个符号的使用状态,从而确定各个可控模块定义的共享符号,其中,可以通过分析可执行文件 exe1 的目标文件 obj1 和目标文件 obj2 来确定可执行文件 exe1 定义的符号。在这里,假设通过 IPA,可以确定函数 l_fun1 和 l_s_

fun1 仅被 lib1 使用, 变量 l_var1 和 l_var2 以及函数 l_fun2 被 lib1 和 lib2 使用, 函数 l_s_fun2 仅被 lib2 使用, 函数 fun1 和 fun2 被 exe1、lib1 和 lib3 使用, 变量 var1 和 var2 以及函数 s_fun1 和 s_fun2 被 exe1 使用。

[0042] 然后, 在步骤 S302 中, 对于可执行文件 exe1, 生成指示可执行文件 exe1 以及可执行文件 exe1 调用的可控模块(即共享库 lib1 和 lib2)定义的共享符号的模块符号使用文件。在本示例中, 可以生成下表 4 所示的表, 作为所述模块符号使用文件。如上所述, 该表可以包括字段“符号”、“Def”和“Ref”三个字段, 其中“符号”字段指示可执行文件 exe1 以及可执行文件 exe1 调用的共享库 lib1 和 lib2 定义的符号, “Def”字段指示定义所述符号的可控模块, “Ref”字段表示使用所述符号的模块。

[0043]

[0044]

符号	Def	Ref
l_fun1	lib1	lib1
l_var1	lib1	lib1, lib2
l_s_fun1	lib1	lib1
l_fun2	lib2	lib1, lib2
l_var2	lib2	lib2, lib1
l_s_fun2	lib2	lib2
fun1	exe1	exe1, lib1, lib3
var1	exe1	exe1
s_fun1	exe1	exe1
fun2	exe1	exe1, lib1, lib3
var2	exe1	exe1
s_fun2	exe1	exe1

[0045] 表 4

[0046] 按照类似的方式, 可以对软件的每个可控模块, 生成指示该可控模块以及该可控模块调用的其他可控模块定义的共享符号的模块符号使用文件。然后, 可以如上文所述, 合并对各个可控模块生成的模块符号使用文件, 以生成指示所述至少一个可控模块中的所有共享符号的总符号使用文件。

[0047] 返回图 3, 在步骤 S303 中, 基于所述总符号使用文件重新构建该软件, 使得重新构建后的软件的可控模块中的符号表仅包含所述总符号使用文件中指示的共享符号。至于内

部符号,由于其只被定义该内部符号的可控模块使用,因此可以在需要时在该可控模块中直接使用它们,而不将其存储在符号表中。

[0048] 具体地,可以对该软件的每个可控模块的源文件进行重新编译以产生相应的目标文件,其中在所述目标文件中,对所述每个可控模块中除了所述总符号使用文件中指示的共享符号以外的符号(即,内部符号)设置比所述共享符号更快的访问方式。更具体地,对于每个可控模块中的共享符号,由于其被多个模块使用,因此需要在该可控模块的目标文件中提供所述共享符号的寻址信息,并且设置所述共享符号的访问方式,使得所述可控模块根据该寻址信息来定位和加载(访问)所述共享符号。对于每个可控模块中的内部符号,由于其仅被该可控模块自己使用,因此可以不在该可控模块的目标文件中设置寻址信息,而是设置所述内部符号的访问方式,使得所述可控模块直接加载(访问)所述内部符号,从而获得比共享符号更快的访问速度。例如,可以将所述总符号使用文件提供给编译器。编译器在对该软件的各个可控模块的源文件进行编译期间,查询所述总符号使用文件以确定各个可控模块中的共享符号(以及内部符号),然后在编译产生的目标文件中,对各个可控模块中的内部符号设置快速访问方式,而对共享符号设置常规的访问方式(慢速访问方式)。这样,可以优化为各个可控模块中生成的目标文件。

[0049] 然后,可以将优化后的目标文件以及所述总符号使用文件提供给链接器。链接器查询所述总符号使用文件以确定各个可控模块中的共享符号(以及内部符号),并且在重新链接所述目标文件以产生重新构建后的软件中分别与所述各个可控模块对应的可控模块(可执行文件或共享库)期间,在所述对应的可控模块中的符号表中隐藏所述内部符号,即,不将所述内部符号生成到重新构建后的软件中的可控模块中的符号表中,使得符号表仅包含所述总符号使用文件指示的共享符号。

[0050] 这样,在需要时,可以运行重新构建后的软件,而不再运行原来的软件。可以看到,利用根据本发明实施例的优化软件的方法,可以自动地分析软件的各个可控模块以确定其定义的各个符号的使用状态,与人工分析所述使用状态的传统方法相比,显著减轻了软件开发者的负担。此外,所述方法可以准确地识别软件中的内部符号并减少符号表中的符号数量,从而减少软件(即,重新构建后的软件)运行时加载和解析该符号表耗费的时间,提高软件的运行效率。而且,由于在重新构建后的软件的各个可控模块的符号表中隐藏了内部符号,因此可以防止软件的实现细节被暴露给外部,提高了软件的安全性。

[0051] 应当认识到,上文所述的优化软件的方法只是说明性的,而不是限制性的,本领域技术人员可以对其做出各种改变,而不背离本发明的范围。例如,尽管在上文中针对步骤S303的描述中提到将整个总符号使用文件提供给编译器和链接器以重新构建所述软件,这不是限制性的,也可以按照定义各个符号的模块(例如,上表4中的“Def”字段中指示的模块),将总符号使用文件划分为多个子文件。然后,在对每个可控模块进行重新编译和链接时,将与该可控子模块对应的子文件提供给编译器和链接器,使得编译器和链接器仅基于该子文件来进行编译和链接。例如,可以按照定义各个符号的模块,将上表3所述的总符号使用文件划分为下表5和6所示的两个子文件:

[0052]

符号	Def	Ref

sym1	mod1	mod1, mod2
------	------	------------

[0053] 表 5

[0054]

符号	Def	Ref
sym2	mod2	mod2, mod1
sym3	mod2	mod2

[0055] 表 6

[0056] 然后,在对可控模块 1 (mod1) 进行重新编译和链接时,可以将表 5 所示的子文件提供给编译器和链接器,而在在对可控模块 2 (mod2) 进行重新编译和链接时,可以将表 6 所示的子文件提供给编译器和链接器。这样,由于编译器和链接器接收的子文件包含的符号数量减少,因此可以减少加载和解析该子文件花费的时间。

[0057] 此外,上述根据本发明实施例的优化软件的方法可以与上文所述的两种传统方法结合使用。例如,在利用根据本发明实施例的优化软件的方法产生了总符号使用文件之后,可以按照预定的合并规则,将开发者自己编写的导出文件合并到该总符号使用文件中,然后利用合并后的总符号使用文件来重新构建所述软件。所述合并规则例如可以是:如果某个符号在导出文件中被标记为“导出”,则无论该符号在合并前的总符号使用文件是否被标记为共享符号,在合并后的总符号使用文件中都将该符号标记为共享符号。例如,可以通过在合并后的总符号使用文件的“Ref”字段中增加相应的标签来进行这一标记。这样,软件的开发者可以通过编写导出文件来灵活地控制是否导出软件中的符号。

[0058] 前面已经参考附图描述了实现本发明的方法的各个实施例。本领域技术人员可以理解的是,上述方法既可以以软件方式实现,也可以以硬件方式实现,或者通过软件与硬件相结合的方式实现。并且,本领域技术人员可以理解,通过以软件、硬件或者软硬件相结合的方式实现上述方法中的各个步骤,可以提供一种基于相同发明构思的优化软件的设备。即使该设备在硬件结构上与通用处理设备相同,由于其中所包含的软件的作用,使得该设备表现出区别于通用处理设备的特性,从而形成本发明的各个实施例的设备。本发明中所述设备包括若干单元或模块,所述单元或模块被配置为执行相应步骤。本领域的所述技术人员通过阅读本说明书可以理解如何编写程序实现所述单元或模块执行的动作。下面,将参考图 5 具体描述根据本发明实施例的优化软件的设备。由于所述设备与所述方法基于相同的发明构思,因此其中相同或相应的实现细节同样适用于与上述方法对应的设备,由于其在上文中已经进行了详细和完整的描述,因此在下文中可能不再进行赘述。

[0059] 如图 5 所示,根据本发明实施例的优化软件的设备(以下可称为优化设备) 500 可以包括确定装置 501、生成装置 502 和构建装置 503。

[0060] 确定装置 501 可以确定所述至少一个可控模块定义的共享符号。如上文所述,所述共享符号是被软件的两个或更多模块(包括可控模块和 / 或非可控模块)使用的符号。具体地,确定装置 501 可以通过分析所述至少一个可控模块中的每个可控模块定义的符号是否被所述每个可控模块和所述每个可控模块调用的其他模块(包括可控模块和 / 或非可控

模块)中的两个或更多模块使用,来确定所述每个可控模块定义的符号是否是共享符号。确定装置 501 可以通过对所述每个可控模块和所述每个可控模块调用的其他模块执行 IPA, 来分析每个可控模块定义的符号的使用状态, 即, 确定每个可控模块定义的符号是否被所述每个可控模块和所述每个可控模块调用的其他模块中的两个或更多模块使用。确定装置 501 执行 IPA 的方法是本领域公知的, 在这里不再赘述。

[0061] 生成模块 502 可以生成指示所述至少一个可控模块定义的共享符号的符号使用文件(即, 上文所述的总符号使用文件)。除了指示所述至少一个可控模块定义的共享符号以外, 所述总符号使用文件还可以包含指示定义所述共享符号的可控模块的信息以及指示使用所述符号的模块的信息。生成模块 502 可以按照在上文中针对图 2 描述的方式来生成所述总符号使用文件, 在这里为了避免重复而省略其详细描述。所述总符号使用文件可以采用任何适当的格式, 例如表格、文本文件等。

[0062] 构建模块 503 可以基于所述总符号使用文件重新构建该软件, 使得重新构建后的软件的可控模块中的符号表仅包含所述符号使用文件中指示的共享符号。

[0063] 图 6 示出了构建模块 503 的结构。如图 6 所示, 构建模块 503 可以包括编译器 5031 和链接器 5032。编译器 5031 可以对该软件的每个可控模块的源文件进行重新编译以产生相应的目标文件, 其中在所述目标文件中, 对所述每个可控模块中除了所述总符号使用文件中指示的共享符号以外的符号(即, 内部符号)设置比所述共享符号更快的访问方式。例如, 编译器 5031 在对该软件的各个可控模块的源文件进行编译期间, 可以查询所述总符号使用文件以确定各个可控模块中的共享符号, 然后在编译产生的目标文件中, 对各个可控模块中的内部符号设置快速访问方式, 而对共享符号设置常规的慢速访问方式。链接器 5032 可以接收优化后的目标文件以及所述总符号使用文件, 查询所述总符号使用文件以确定各个可控模块中的共享符号(以及内部符号), 并且在重新链接所述目标文件以产生重新构建后的软件中与所述各个可控模块对应的可控模块(可执行文件或共享库)期间, 在所述对应的可控模块的符号表中隐藏所述内部符号, 即, 不将所述内部符号生成到重新构建后的软件中的可控模块的符号表中, 使得所述可控模块的符号表仅包含所述总符号使用文件指示的共享符号。

[0064] 这样, 在需要时, 可以运行重新构建后的软件, 而不再运行原来的软件。可以看到, 利用根据本发明实施例的优化软件的设备, 可以准确地识别软件中的内部符号和共享符号以减少符号表中的符号数量, 从而减少软件(即, 重新构建后的软件)运行时加载和解析该符号表所耗费的时间, 提高软件的运行效率。而且, 由于在重新构建后的软件中的可控模块的符号表中隐藏了内部符号, 因此可以防止该软件的实现细节被暴露给外部, 提高了软件的安全性。

[0065] 本发明可以是系统、方法和 / 或计算机程序产品。计算机程序产品可以包括计算机可读存储介质, 其上载有用于使处理器实现本发明的各个方面的计算机可读程序指令。

[0066] 计算机可读存储介质可以是可以保持和存储由指令执行设备使用的指令的有形设备。计算机可读存储介质例如可以是一一但不限于一一电存储设备、磁存储设备、光存储设备、电磁存储设备、半导体存储设备或者上述的任意合适的组合。计算机可读存储介质的更具体的例子(非穷举的列表)包括:便携式计算机盘、硬盘、随机存取存储器(RAM)、只读存储器(ROM)、可擦式可编程只读存储器(EPROM 或闪存)、静态随机存取存储器(SRAM)、便携

式压缩盘只读存储器(CD-ROM)、数字多功能盘(DVD)、记忆棒、软盘、机械编码设备、例如其上存储有指令的打孔卡或凹槽内凸起结构、以及上述的任意合适的组合。这里所使用的计算机可读存储介质不被解释为瞬时信号本身，诸如无线电波或者其他自由传播的电磁波、通过波导或其他传输媒介传播的电磁波(例如，通过光纤电缆的光脉冲)、或者通过电线传输的电信号。

[0067] 这里所描述的计算机可读程序指令可以从计算机可读存储介质下载到各个计算/处理设备，或者通过网络、例如因特网、局域网、广域网和/或无线网下载到外部计算机或外部存储设备。网络可以包括铜传输电缆、光纤传输、无线传输、路由器、防火墙、交换机、网关计算机和/或边缘服务器。每个计算/处理设备中的网络适配卡或者网络接口从网络接收计算机可读程序指令，并转发该计算机可读程序指令，以供存储在各个计算/处理设备中的计算机可读存储介质中。

[0068] 用于执行本发明操作的计算机程序指令可以是汇编指令、指令集架构(ISA)指令、机器指令、机器相关指令、微代码、固件指令、状态设置数据、或者以一种或多种编程语言的任意组合编写的源代码或目标代码，所述编程语言包括面向对象的编程语言—诸如Smalltalk、C++等，以及常规的过程式编程语言—诸如“C”语言或类似的编程语言。计算机可读程序指令可以完全地在用户计算机上执行、部分地在用户计算机上执行、作为一个独立的软件包执行、部分在用户计算机上部分在远程计算机上执行、或者完全在远程计算机或服务器上执行。在涉及远程计算机的情形中，远程计算机可以通过任意种类的网络—包括局域网(LAN)或广域网(WAN)—连接到用户计算机，或者，可以连接到外部计算机(例如利用因特网服务提供商来通过因特网连接)。在一些实施例中，通过利用计算机可读程序指令的状态信息来个性化定制电子电路，例如可编程逻辑电路、现场可编程门阵列(FPGA)或可编程逻辑阵列(PLA)，该电子电路可以执行计算机可读程序指令，从而实现本发明的各个方面。

[0069] 这里参照根据本发明实施例的方法、装置(系统)和计算机程序产品的流程图和/或框图描述了本发明的各个方面。应当理解，流程图和/或框图的每个方框以及流程图和/或框图中各方框的组合，都可以由计算机可读程序指令实现。

[0070] 这些计算机可读程序指令可以提供给通用计算机、专用计算机或其它可编程数据处理装置的处理器，从而生产出一种机器，使得这些指令在通过计算机或其它可编程数据处理装置的处理器执行时，产生了实现流程图和/或框图中的一个或多个方框中规定的功能/动作的装置。也可以把这些计算机可读程序指令存储在计算机可读存储介质中，这些指令使得计算机、可编程数据处理装置和/或其他设备以特定方式工作，从而，存储有指令的计算机可读介质则包括一个制造品，其包括实现流程图和/或框图中的一个或多个方框中规定的功能/动作的各个方面指令。

[0071] 也可以把计算机可读程序指令加载到计算机、其它可编程数据处理装置、或其它设备上，使得在计算机、其它可编程数据处理装置或其它设备上执行一系列操作步骤，以产生计算机实现的过程，从而使得在计算机、其它可编程数据处理装置、或其它设备上执行的指令实现流程图和/或框图中的一个或多个方框中规定的功能/动作。

[0072] 附图中的流程图和框图显示了根据本发明的多个实施例的系统、方法和计算机程序产品的可能实现的体系架构、功能和操作。在这点上，流程图或框图中的每个方框可以

代表一个模块、程序段或指令的一部分，所述模块、程序段或指令的一部分包含一个或多个用于实现规定的逻辑功能的可执行指令。在有些作为替换的实现中，方框中所标注的功能也可以以不同于附图中所标注的顺序发生。例如，两个连续的方框实际上可以基本并行地执行，它们有时也可以按相反的顺序执行，这依所涉及的功能而定。也要注意的是，框图和 / 或流程图中的每个方框、以及框图和 / 或流程图中的方框的组合，可以用执行规定的功能或动作的专用的基于硬件的系统来实现，或者可以用专用硬件与计算机指令的组合来实现。

[0073] 以上已经描述了本发明的各实施例，上述说明是示例性的，并非穷尽性的，并且也不限于所披露的各实施例。在不偏离所说明的各实施例的范围和精神的情况下，对于本技术领域的普通技术人员来说许多修改和变更都是显而易见的。本文中所用术语的选择，旨在最好地解释各实施例的原理、实际应用或对市场中的技术的技术改进，或者使本技术领域的其它普通技术人员能理解本文披露的各实施例。

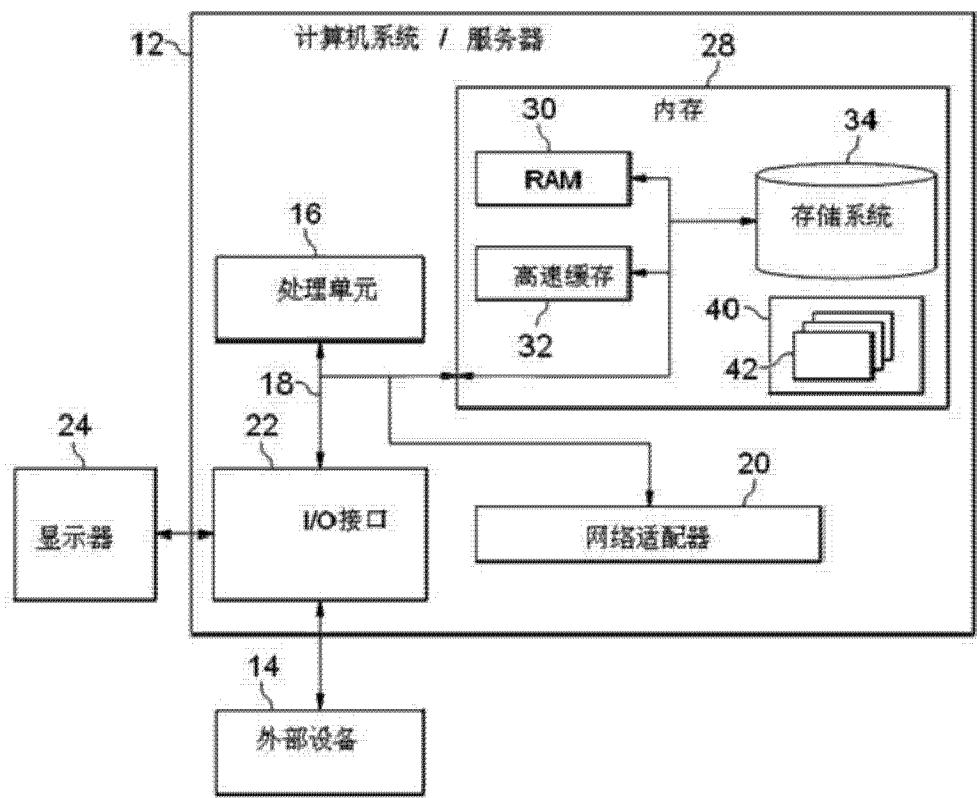


图 1

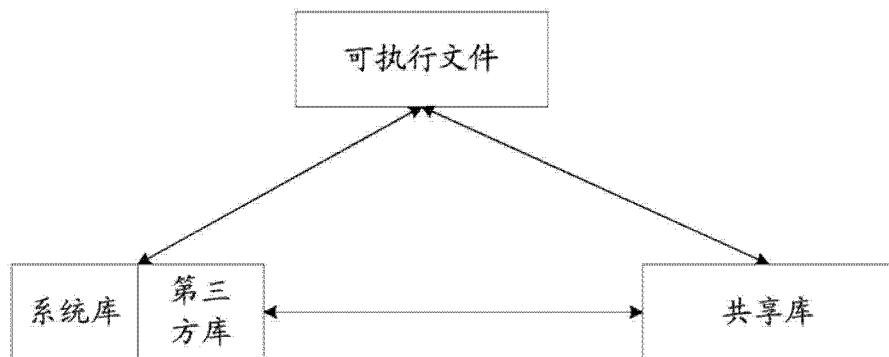


图 2

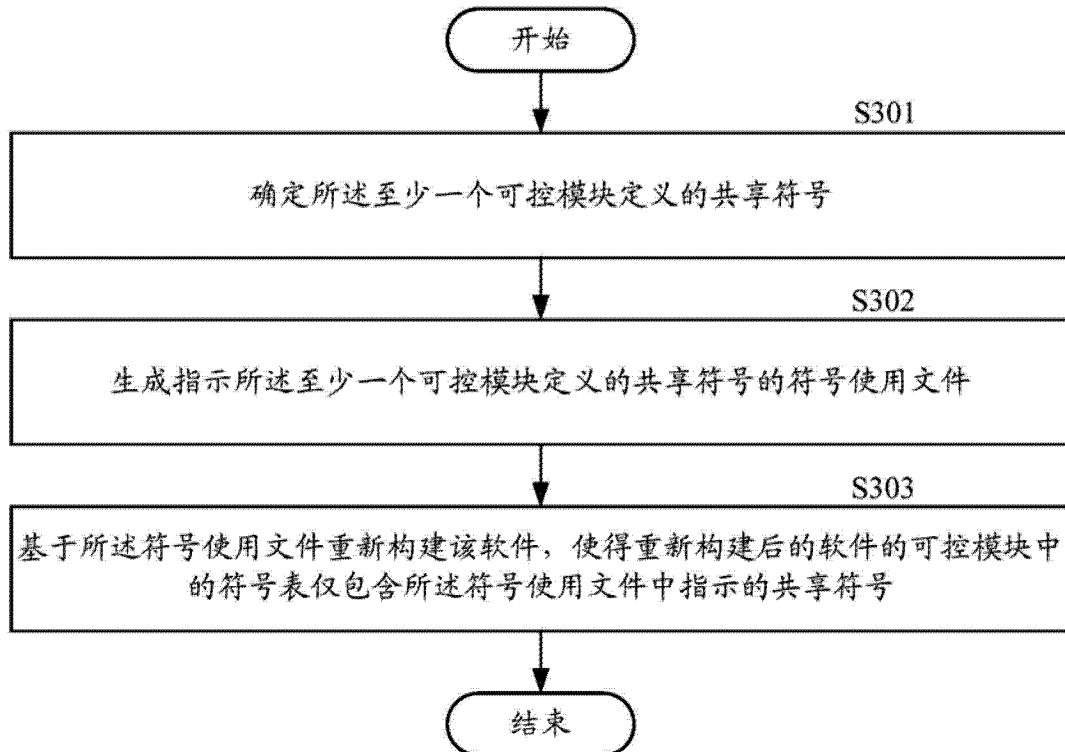


图 3

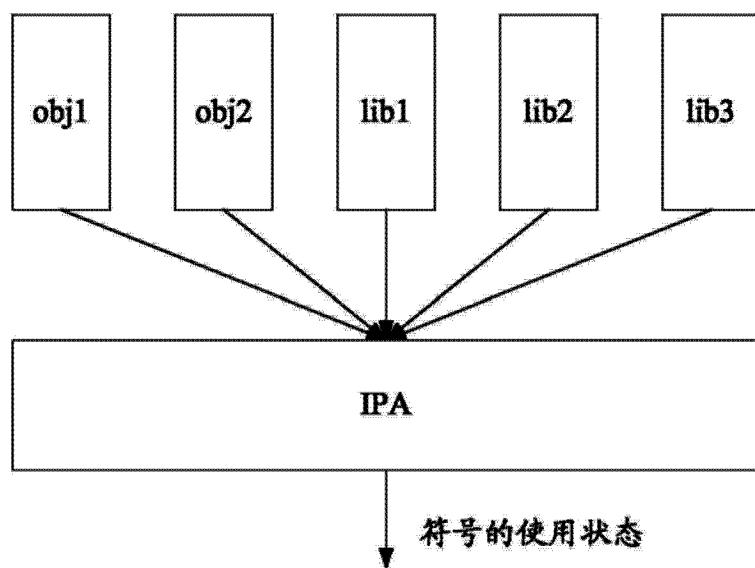


图 4

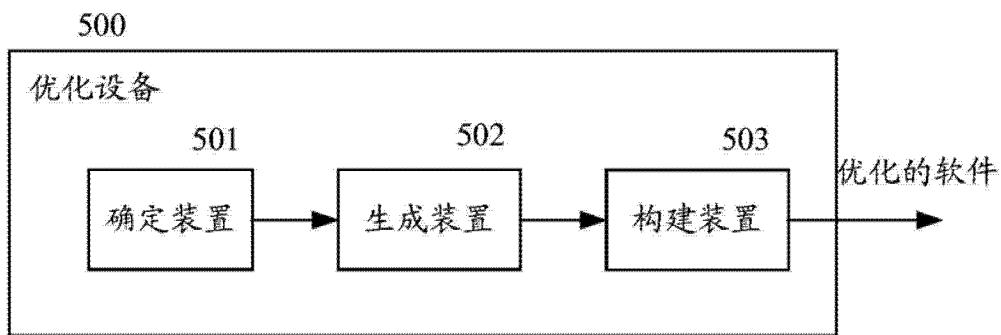


图 5

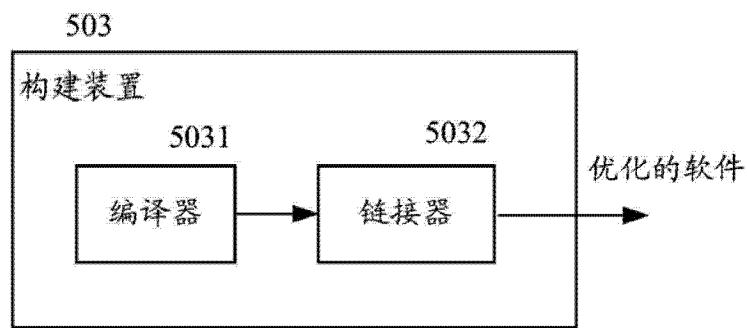


图 6