(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 116661779 B
(45)授权公告日 2023.09.19

(21)申请号 202310926311.X

(22)申请日 2023.07.26

(65)同一申请的已公布的文献号
申请公布号 CN 116661779 A

(43)申请公布日 2023.08.29

(73)专利权人 北京麟卓信息科技有限公司
地址 100085 北京市海淀区西三旗昌临801号27号3层310、312

(72)发明人 温研 杜凯

(51)Int.Cl.
G06F 8/36(2018.01)
G06F 8/41(2018.01)
G06F 9/445(2018.01)
G06F 9/448(2018.01)

(56)对比文件
CN 103729579 A,2014.04.16
CN 105426226 A,2016.03.23
CN 107239315 A,2017.10.10
CN 108052327 A,2018.05.18
CN 114815990 A,2022.07.29
CN 116400895 A,2023.07.07
US 2019171426 A1,2019.06.06

审查员 王宇莉

权利要求书2页 说明书6页

(54)发明名称

一种基于符号动态重定向的多编译器混合链接方法

(57)摘要

本发明公开了一种基于符号动态重定向的多编译器混合链接方法,通过基于构建的编译器与其符号还原功能间的映射关系得到错误符号在源代码中的函数名或变量名,同时获取当前使用的所有链接库中符号在源代码中的函数名或变量名构成库源名称表,通过与库源名称表比对确定错误符号对应的符号,再通过构造封装库完成错误符号的代理及导出,从而解决了由编译器名称修饰规则差异所导致的程序编译时符号不匹配问题。

CN 116661779 B

1.一种基于符号动态重定向的多编译器混合链接方法,其特征在于,包括以下步骤:

步骤1、提取不同编译器及同一编译器的不同版本的符号还原功能,将符号还原功能封装为API接口记为符号还原API,建立编译器与符号还原API间的映射关系记为编译器还原映射;

步骤2、若第一编译器编译目标程序执行链接操作产生无法解析符号错误,则将当前错误符号加入错误符号列表,根据链接顺序依次将当前链接操作相关的链接库加入链接库列表;否则,若目标程序已全部完成编译链接则结束本流程,若目标程序未全部完成编译链接则执行步骤2;

步骤3、在编译器还原映射中查找第一编译器对应的符号还原API记为第一符号还原API,采用第一符号还原API对当前错误符号执行符号还原操作得到其在源代码中的函数名或变量名记为第一源名称;

步骤4、将生成链接库列表中链接库的编译器的名称记为第二编译器,在编译器还原映射中查找第二编译器对应的符号还原API记为第二符号还原API;采用第二符号还原API对链接库的所有符号执行符号还原操作得到其在源代码中的函数名或变量名,将这些函数名或变量名加入库源名称表;

步骤5、若库源名称表中存在与第一源名称相同的名称,则将与第一源名称相匹配的符号在其对应的链接库中的符号记为第一库符号,与其对应的链接库记为第一链接库;建立第一封装库封装第一代理的源代码,第一代理为名称与第一源名称相同的当前错误符号的代理函数或代理变量,将第一代理的标识设置为已导出;若不存在,则结束本流程;

步骤6、在第一封装库中加入初始化函数记为封装初始化函数,再将封装初始化函数加入目标程序的初始化函数列表,使目标程序执行时即可调用封装初始化函数;封装初始化函数用于处理当前错误符号为变量时的数据访问异常,当产生异常时获取第一代理的内存地址作为新的指令地址执行后续操作;

步骤7、采用第一编译器编译第一封装库得到第一封装链接库,将第一封装链接库添加到链接库列表,执行步骤2。

2.根据权利要求1所述的多编译器混合链接方法,其特征在于,所述链接库为静态库或动态库。

3.根据权利要求1所述的多编译器混合链接方法,其特征在于,所述步骤7中所述第一封装链接库为静态库或动态库。

4.根据权利要求2所述的多编译器混合链接方法,其特征在于,还包括当前错误符号为函数时所述第一代理的实现方式为:

第一代理为与第一源名称相同的函数,若第一链接库为动态库,则调用动态库加载函数加载第一链接库,再调用符号动态解析函数获取第一库符号的地址记为函数地址,根据函数地址调用第一库符号;若第一链接库为静态库,则获取第一库符号在第一链接库中的相对地址记为偏移地址,获取第一链接库的地址记为基地址,将偏移地址与基地址的和作为函数地址,跳转到该函数地址;

将第一代理的源代码加入到第一封装库的源代码中。

5.根据权利要求2所述的多编译器混合链接方法,其特征在于,还包括当前错误符号为变量时所述第一代理的实现方式为:

第一代理为与第一源名称相同的变量,若为首次执行,则在第一封装库中添加不可读写的数据段记为标识数据段,再将第一代理定义为数据结构,所述数据结构包含第一链接库的地址即基地址和第一库符号在第一链接库中的相对地址即偏移地址,再将第一代理写入标识数据段中,将第一代理的源代码加入到第一封装库的源代码中;否则将第一代理的源代码加入到第一封装库的源代码中。

6.根据权利要求1所述的多编译器混合链接方法,其特征在于,所述步骤6中所述封装初始化函数的实现方式为:

当捕获到的数据访问异常中的数据地址处于标识数据段中时,获取标识数据段中的数据结构,将数据结构中的基地址与偏移地址求和得到地址A,用地址A替换触发当前异常的指令A的操作数地址,再将当前进程的指令地址指向指令A后继续执行当前进程。

7.根据权利要求1所述的多编译器混合链接方法,其特征在于,所述编译器还原映射中的编译器包含编译器的名称和版本号。

一种基于符号动态重定向的多编译器混合链接方法

技术领域

[0001] 本发明属于计算机应用开发技术领域,具体涉及一种基于符号动态重定向的多编译器混合链接方法。

背景技术

[0002] 在某些编程语言中函数重载或方法重载主要用于创建多个具有不同实现的同名函数。调用重载函数时将运行适合于调用上下文的该函数的特定实现,换句话说,调用重载函数能够实现根据上下文执行不同的任务。

[0003] 在编译器构造中名称修饰(Name Mangling)用于解决为编程实体解析唯一名称而引起的各种问题,它提供了一种以函数、结构、类或其他数据类型的名称对附加信息进行编码的方法,以便将更多语义信息从编译器传递至链接器(Linker)。但是,不同编译器,甚至同一编译器的不同版本,其名称修饰规则并不相同,也就是说同一函数经不同编译器编译后的名称往往会存在较大差异,这将导致由某个编译器生成的动态库或静态库无法被其他编译器识别,也就无法被在编译程序源码后链接时使用。例如,由GCC编译动态库或静态库libA,libA实现并导出了函数fun,GCC将函数fun名称修饰后导出的符号是funX,然而若用户编写程序A使用函数fun并采用clang编译器执行编译时,由于按照clang的名称修饰规则其将函数fun名称修饰为funY,此时就会导致程序A链接失败。

[0004] 综上所述,现有编译框架存在由不同编译器或同一编译器的不同版本间名称修饰规则差异所导致的程序编译时符号不匹配的问题。

发明内容

[0005] 有鉴于此,本发明提供了一种基于符号动态重定向的多编译器混合链接方法,实现了符号在多编译器下的混合链接。

[0006] 本发明提供的一种基于符号动态重定向的多编译器混合链接方法,包括以下步骤:

[0007] 步骤1、提取不同编译器及同一编译器的不同版本的符号还原功能,将符号还原功能封装为API接口记为符号还原API,建立编译器与符号还原API间的映射关系记为编译器还原映射;

[0008] 步骤2、若第一编译器编译目标程序执行链接操作产生无法解析符号错误,则将当前错误符号加入错误符号列表,根据链接顺序依次将当前链接操作相关的链接库加入链接库列表;否则,若目标程序已全部完成编译链接则结束本流程,若目标程序未全部完成编译链接则执行步骤2;

[0009] 步骤3、在编译器还原映射中查找第一编译器对应的符号还原API记为第一符号还原API,采用第一符号还原API对当前错误符号执行符号还原操作得到其在源代码中的函数名或变量名记为第一源名称;

[0010] 步骤4、将生成链接库列表中链接库的编译器的名称记为第二编译器,在编译器还

原映射中查找第二编译器对应的符号还原API记为第二符号还原API;采用第二符号还原API对链接库的所有符号执行符号还原操作得到其在源代码中的函数名或变量名,将这些函数名或变量名加入库源名称表;

[0011] 步骤5、若库源名称表中存在与第一源名称相同的名称,则将与第一源名称相匹配的符号在其对应的链接库中的符号记为第一库符号,与其对应的链接库记为第一链接库;建立第一封装库封装第一代理的源代码,第一代理为名称与第一源名称相同的当前错误符号的代理函数或代理变量,将第一代理的标识设置为已导出;若不存在,则结束本流程;

[0012] 步骤6、在第一封装库中加入初始化函数记为封装初始化函数,再将封装初始化函数加入目标程序的初始化函数列表,使目标程序执行时即可调用封装初始化函数;封装初始化函数用于处理当前错误符号为变量时的数据访问异常,当产生异常时获取第一代理的内存地址作为新的指令地址执行后续操作;

[0013] 步骤7、采用第一编译器编译第一封装库得到第一封装链接库,将第一封装链接库添加到链接库列表,执行步骤2。

[0014] 进一步地,所述链接库为静态库或动态库。

[0015] 进一步地,所述步骤7中所述第一封装链接库为静态库或动态库。

[0016] 进一步地,还包括当前错误符号为函数时所述第一代理的实现方式为:

[0017] 第一代理为与第一源名称相同的函数,若第一链接库为动态库,则调用动态库加载函数加载第一链接库,再调用符号动态解析函数获取第一库符号的地址记为函数地址,根据函数地址调用第一库符号;若第一链接库为静态库,则获取第一库符号在第一链接库中的相对地址记为偏移地址,获取第一链接库的地址记为基地址,将偏移地址与基地址的和作为函数地址,跳转到该函数地址;

[0018] 将第一代理的源代码加入到第一封装库的源代码中。

[0019] 进一步地,还包括当前错误符号为变量时所述第一代理的实现方式为:

[0020] 第一代理为与第一源名称相同的变量,若为首次执行,则在第一封装库中添加不可读写的数据段记为标识数据段,再将第一代理定义为数据结构,所述数据结构包含第一链接库的地址即基地址和第一库符号在第一链接库中的相对地址即偏移地址,再将第一代理写入标识数据段中,将第一代理的源代码加入到第一封装库的源代码中;否则将第一代理的源代码加入到第一封装库的源代码中。

[0021] 进一步地,所述步骤6中所述封装初始化函数的实现方式为:

[0022] 当捕获到的数据访问异常中的数据地址处于标识数据段中时,获取标识数据段中的数据结构,将数据结构中的基地址与偏移地址求和得到地址A,用地址A替换触发当前异常的指令A的操作数地址,再将当前进程的指令地址指向指令A后继续执行当前进程。

[0023] 进一步地,所述编译器还原映射中的编译器包含编译器的名称和版本号。

[0024] 有益效果

[0025] 本发明通过基于构建的编译器与其符号还原功能间的映射关系得到错误符号在源代码中的函数名或变量名,同时获取当前使用的所有链接库中符号在源代码中的函数名或变量名构成库源名称表,通过与库源名称表比对确定错误符号对应的符号,再通过构造封装库完成错误符号的代理及导出,从而解决了由编译器名称修饰规则差异所导致的程序编译时符号不匹配问题。

具体实施方式

[0026] 下面列举实施例,对本发明进行详细描述。

[0027] 本发明提供了一种基于符号动态重定向的多编译器混合链接方法,其核心思想是:基于构建的不同编译器与其符号还原功能间的映射关系,能够得到无法解析符号在源代码中的函数名或变量名,同时获取当前使用的所有链接库中符号在源代码中的函数名或变量名构成库源名称表,通过与库源名称表比对确定错误符号真正对应的符号,再通过构造封装库完成符号的导出。

[0028] 本发明提供的一种基于符号动态重定向的多编译器混合链接方法,具体包括以下步骤:

[0029] 步骤1、针对不同编译器及同一编译器的不同版本,分别提取其符号还原功能,即demangling功能,并将符号还原功能封装为API接口,记为符号还原API;建立不同编译器以及同一编译器的不同版本与符号还原API之间的映射关系,记为编译器还原映射。

[0030] 步骤2、若第一编译器编译目标程序执行链接操作产生无法解析符号错误,则将当前错误符号加入错误符号列表,获取当前链接操作所搜索的链接库,该链接库包括静态库及动态库,并按照链接顺序依次将链接库加入链接库列表;否则,若目标程序已全部完成编译链接则结束本流程,若目标程序未全部完成编译链接则执行步骤2。

[0031] 其中,当前错误符号即为无法解析的符号,该符号的名称是经编译器执行名称修饰后的名称。本发明中,将当前执行程序编译操作的编译器记为第一编译器。

[0032] 步骤3、根据第一编译器的名称及版本在编译器还原映射中查找其对应的符号还原API,记为第一符号还原API;采用第一符号还原API对当前错误符号执行符号还原操作得到其在源代码中的函数名或变量名,记为第一源名称。

[0033] 步骤4、遍历链接库列表中的链接库,获取生成链接库的编译器名称记为第二编译器,根据第二编译器的名称及版本在编译器还原映射中查找其对应的符号还原API,记为第二符号还原API;采用第二符号还原API对链接库的所有符号执行符号还原操作得到其在源代码中的函数名或变量名,将这些函数名或变量名加入库源名称表。

[0034] 步骤5、若库源名称表中存在与第一源名称相同的名称,说明这两个相匹配的符号经过不同编译器或同一编译器的不同版本的名称修饰操作后得到的符号名不同,但其源名称即函数名或变量名是相同的,则将与第一源名称相匹配的符号在其对应的链接库中的符号记为第一库符号,与其对应的链接库记为第一链接库;构造第一封装库用于封装作为当前错误符号的代理函数或代理变量的第一代理的源代码,第一代理的名称与第一源名称相同,将第一代理的标识设置为已导出;若不存在,则结束本流程。

[0035] 其中,第一代理用于获取当前错误符号的实际内存地址,再对实际内存地址的内容执行链接操作。

[0036] 第一代理的一种实现方式为:

[0037] 步骤5.1、若当前错误符号为函数则执行步骤5.2,若当前错误符号为变量则执行步骤5.3。

[0038] 步骤5.2、第一代理为与第一源名称相同的函数,若第一链接库为动态库,则调用动态库加载函数加载第一链接库,再调用符号动态解析函数获取第一库符号的地址记为函数地址,根据函数地址调用第一库符号;若第一链接库为静态库,则获取第一库符号在第一

链接库中的相对地址记为偏移地址，获取第一链接库的地址记为基地址，将偏移地址与基地址的和作为函数地址，跳转到该函数地址，再执行步骤5.4。

[0039] 步骤5.3、第一代理为与第一源名称相同的变量，若为首次执行，则通过增加编译修饰符实现在第一封装库中添加不可读写的数据段记为标识数据段，再将第一代理定义为数据结构，该数据结构包含两个地址类成员，分别为第一链接库的地址即基地址和第一库符号在第一链接库中的相对地址即偏移地址，再通过增加编译修饰符实现将第一代理写入标识数据段中，执行步骤5.4；否则执行步骤5.4。

[0040] 步骤5.4、将第一代理的源代码加入到第一封装库的源代码中，并将第一代理标识为已导出，结束本流程。

[0041] 在源代码中定义函数或变量的位置增加标识符以导出其对应的符号。

[0042] 步骤6、在第一封装库中加入初始化函数记为封装初始化函数，再将封装初始化函数加入目标程序的初始化函数列表，使目标程序执行时即可调用封装初始化函数；封装初始化函数用于处理当前错误符号为变量时的数据访问异常，当产生异常时获取第一代理的内存地址作为新的指令地址执行后续操作。

[0043] 其中，封装初始化函数的一种实现方式为：

[0044] 当捕获到的数据访问异常中的数据地址处于标识数据段中时，获取标识数据段中的数据结构，将数据结构中的基地址与偏移地址求和得到地址A，用地址A替换触发当前异常的指令A的操作数地址，实现将指令A对数据的操作指向第一链接库中的第一代理；再将当前进程的指令地址指向指令A后继续执行。

[0045] 步骤7、采用第一编译器编译第一封装库得到第一封装链接库，将第一封装链接库添加到链接库列表，再执行步骤2。

实施例

[0046] 本实施例采用本发明提供的一种基于符号动态重定向的多编译器混合链接方法实现了 ，具体包括以下步骤：

[0047] S1、针对各编译器的各个版本，分别执行以下步骤：

[0048] 记当前编译器为Compiler，当前版本为V；

[0049] 提取编译器Compiler的版本V的demangling功能，将demangling功能封装为API接口，将该API接口记为demCompilerV，其中Compiler为编译器名称，V为编译器的版本号。

[0050] S2、建立编译器Compiler的版本V与demCompilerV的映射关系，记为mapDemAPI2CompilerV。

[0051] S3、当编译程序programA执行链接操作时出现无法解析符号的错误时，将该无法解析符号加入的列表unresolvedSymbolList中，这里的符号均为经过名称修饰处理过的符号，则：

[0052] S3.1、获取链接programA时链接器Linker搜索的所有动态库和静态库，并按照链接顺序将这些动态库和静态库分别依次加入linkLibList。

[0053] 其中，获取链接programA时链接器Linker搜索的所有动态库和静态库的方式有多种，如从编译命令行中获取、从Makefile中获取或从当前系统链接默认配置中获取等。

[0054] S3.2、将unresolvedSymbolList中每个无法解析的符号记为unresolvedSymbol，

unresolvedSymbol为经过名称修饰的符号,对unresolvedSymbol分别执行以下操作:

[0055] S3.2.1、获取当前编译器的名称和版本,在mapDemAPI2CompilerV中查找到与之对应的demangling功能的API接口demCompilerV。

[0056] S3.2.2、调用demCompilerV对unresolvedSymbol执行demangling操作,获取其被名称修饰之前的源码级的符号名,即其在源代码中所使用的函数名或变量名。

[0057] S3.2.3、对于linkLibList中的每个动态库或静态库,顺序执行以下操作:

[0058] (1)将当前链接库记为linkLib,该链接库为静态库或动态库;

[0059] (2)从linkLib库文件最开始的若干字节的元信息中可获取生成linkLib所采用编译器的名称和版本,在mapDemAPI2CompilerV查找到与其对应的demangling功能API接口,将其记为demCompilerV1;

[0060] (3)使用demCompilerV1对linkLib中所有可链接或可导出的符号执行demangling操作,获取这些符号在被名称修饰之前的源码级的符号名,即函数名或变量名,将源码级的符号名加入源码级符号列表。

[0061] S3.2.4、将unresolvedSymbol在上一步获取的源码级符号列表中进行匹配,若匹配到相同的名称,说明两者虽然被名称修饰后的符号名不同,但在被名称修饰之前的源码级的符号是相同的,则将在linkLib中的符号记为symobolInLinkLib,symobolInLinkLib是经名称修饰后的符号,构造或扩充一个封装库libWrapper,在libWrapper中为unresolvedSymbol生成代理函数或代理变量的源代码,该代理函数或代理变量的源码级名称与unresolvedSymbol被名称修饰之前的名称一致,以保证该符号经同一编译器编译后代理函数或代理变量被名称修饰后的名称也为unresolvedSymbol,具体步骤如下:

[0062] (1)若unresolvedSymbol为函数,则将unresolvedSymbol经名称修饰前的函数名记为demFunction,并为其生成代理函数的源代码,其函数名为demFunction,demFunction的具体实现如下:

[0063] 若linkLib为动态库,则为demFunction生成如下代码:调用动态库加载函数,如Linux系统的dlopen函数,加载linkLib;调用符号动态解析函数,如Linux系统的dlsym函数,获取symobolInLinkLib的地址记为funAddress;根据函数地址funAddress调用symobolInLinkLib;

[0064] 若linkLib为静态库,则获取symobolInLinkLib在linkLib中的相对地址,也就是偏移地址,记为symobolInLinkLibOffset;为demFunction生成如下代码:获取linkLib的基地址,记为linkLibBaseAddress,使用跳转指令,如x86系统的jmp指令,跳转到地址linkLibBaseAddress与symobolInLinkLibOffset求和的地址;

[0065] 将demFunction代码添加到libWrapper的源代码中;

[0066] 将demFunction标识为已导出。

[0067] (2)若unresolvedSymbol为变量,则将unresolvedSymbol经名称修饰前的变量名记为demVar,为其生成代理变量的源代码,则:

[0068] 若为第一次执行,则通过添加编译修饰符在libWrapper加入不可读写的数据段dataSection,将demVar定义为数据结构,该数据结构包含两个地址类成员包括linkLib的基地址及symobolInLinkLib在linkLib中的偏移地址,再通过添加编译修饰符将demVar加入dataSection;

[0069] 若不是第一次执行,则将demVar标识为已导出。

[0070] S4、在libWrapper中加入初始化函数,记为libWrapperInitFuntion,将libWrapperInitFuntion加入到programA的初始化函数列表中,通过增加编译选项实现,这样programA执行时可调用libWrapperInitFuntion。

[0071] libWrapperInitFuntion的具体实现如下:

[0072] 为当前进程加入数据访问异常处理函数,该函数的实现为:若捕获到的数据访问异常相关的数据地址处于dataSection中,则从该地址中获取对应的数据结构,即包括linkLib基地址和变量的偏移地址,采用二者求和后的结果替换触发当前异常的指令的操作数地址,以实现令当前异常的指令的数据操作指向linkLib中的demVar;将当前进程的指令地址指向发生异常的指令,恢复异常后继续执行当前进程。

[0073] S5、用当前编译器编译libWrapper,生成链接库,该链接库可为静态库或动态库。

[0074] S6、将libWrapper的链接库加入到programA的链接库列表中,重新编译链接programA。

[0075] 综上所述,以上仅为本发明的较佳实施例而已,并非用于限定本发明的保护范围。凡在本发明的精神和原则之内,所作的任何修改、等同替换、改进等,均应包含在本发明的保护范围之内。