(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 103034486 A
(43)申请公布日 2013.04.10

(21)申请号 201210495453.7

(22)申请日 2012.11.28

(71)申请人 清华大学
地址 100084 北京市海淀区清华园 1 号

(72)发明人 董渊 王生原 李叠 骆欢

(74)专利代理机构 北京思海天达知识产权代理有限公司 11203
代理人 楼艮基

(51)Int.Cl.
G06F 9/44(2006.01)

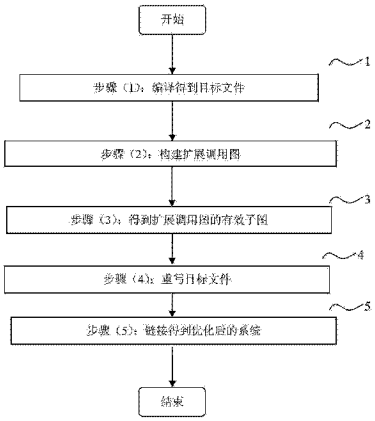
权利要求书 2 页 说明书 7 页 附图 1 页

(54)发明名称

移动终端操作系统基于全系统扩展调用图的自动优化方法

(57)摘要

一种移动终端操作系统基于全系统扩展调用图的自动优化方法，涉及移动终端软件领域，所述方法适用于移动终端操作系统，可以实现在全系统范围内消除无用代码，在不降低系统性能的前提下减小系统代码体积，从而减少移动终端硬件制造成本。方法的主要步骤如下：编译源代码得到目标文件；从目标文件出发，构建全系统所有本地代码的扩展调用图；对扩展调用图进行分析，从入口点开始得到扩展调用图的有效子图；对编译生成的目标文件进行重写，删除所有不在有效子图中的代码；链接生成优化的二进制代码。本发明具有应用范围广、自动化程度高、使用方便、优化效果明显等特点。



CN 103034486 A

1. 一种移动终端操作系统基于全系统扩展调用图的自动优化方法，其特征在于，是在计算机中依次按以下步骤实现的：

步骤(1)：编译移动终端操作系统的源代码，生成可重定位的目标文件；

从源代码服务器中下载源代码，修改相关的Makefile，向CFLAGS添加“-ffunction-sections”参数和“-fdata-sections”参数以使编译器将每个函数和数据对象编译到单独的段中，再运行make命令自动编译所述源代码生成可重定位的目标文件，其中函数和数据对象统称为“实体”，所述数据对象既包括源代码中定义的全局和静态变量，也至少包括虚函数表在内的编译器生成的数据对象；

步骤(2)：按以下步骤构造有向的全系统扩展调用图：

步骤(2.1)：对步骤(1)得到的目标文件，读取以下信息，其中包括SS、GS、AF和AU：

SS，所述目标文件中所有包含有所述实体的段的名称集合，

GS，所述目标文件中定义的所有全局符号及其名称，并以关联表的形式存储，以便从全局符号的名称迅速查找所在的段的名称，

AF，头和尾都属于同一个所述目标文件的有向边形成的集合，

AU，头属于一个所述目标文件，尾为暂未解析的符号的有向边，其中每一个元素表示为(u,sym)，其中u属于SS，是所述目标文件中的一个所述实体，sym为一个外部符号的名，表示被实体u引用，但不在同一个所述目标文件中定义的实体，

步骤(2.2)，根据步骤(2.1)中所得到的信息，合并为一个系统全局的有向图，从而得到所述扩展调用图的V和E，其中：

V为所有所述目标文件SS的并集，是一个结点集，其中每一个结点的名称用所对应的一个二元组表示，其中包括目标文件名和段名，

E为有向边集，它包括两部分：$E=E_1 \cup E_2$，其中E_1为所有所述目标文件的AF的并集，代表了头和尾都属于同一个所述目标文件的有向边所形成的集合，E_2为头和尾属于不同的所述目标文件的有向边形成的集合，按以下方法得到：首先，令$E_2=\varnothing$，再遍历每个所述目标文件的AU集合，对其中的每一个元素(u，sym)，查找所有所述目标文件的GS集，得到所有名称为sym的全局符号的所述实体集合S[sym]，将所有二元组$(u,v) \mid _{v \in S[sym]}$加入集合$E_2$，并将$E_1$与$E_2$取并集得到E，其中每一个元素是一对结点(u,v)，对应所述系统扩展调用图中一条从u到v的有向边，当且仅当u的重定位数据中有相对v的重定位记录时，在所述系统扩展调用图中存在一条从u到v的边，

步骤(2.3)，按以下步骤得到入口点集R：

步骤(2.3.1)，令$R=\varnothing$，

步骤(2.3.2)，把同程序启动代码对应的所述实体作为一个结点加入所述入口点集R，在Android中_start符号所对应的实体即为程序启动代码，

步骤(2.3.3)，把可能通过动态绑定使用的所述实体作为一个结点加入所述入口点集R，

步骤(2.3.4)，把位于不以.text、.data、.rodata、.bss打头的段中的所述实体作为一个结点加入所述入口点集R，

步骤(2.4)，从步骤(2.1)至步骤(2.3)得到的V、E、R表示为一个所述的系统调用图G=(V,E,R)；

步骤(3),按以下步骤从步骤(2)得到的所述系统扩展调用图G中得到有效子图$G_S=(V_S,E_S,R)$,其中V_S为有效结点集,E_S为有效的有向边集,表示为:

$$V_S=\bigcup_{u\in R} Desc\ (u)$$

$$E_S=E\cap(V_S\times V_S)$$

其中Desc(u)表示u在扩展调用图G中包括u自身在内的子孙结点的集合,

步骤(3.1),令$V_S=R$,$Q=R$表示待访问的结点集,$VisitedV=\varnothing$表示已访问的结点集,

步骤(3.2),从待访问的结点集Q中任取一个结点u加入VisitedV,再把结点u在所述系统扩展调用图G中的所有直接后继结点加入所述V_S中,把述结点u的不属于所述已访问过的结点集VisitedV的直接后继结点加入待访问的结点集Q中,并从所述待访问的结点集Q中删除访问过的结点u,

步骤(3.3),重复步骤(3.2),直到所述待访问的结点集Q为空,得到有效结点集V_S,

步骤(3.4),令E_S为空集,遍历E中的所有有向边(u,v),若结点u和结点v都属于所述有效结点集V_S,把从所述结点u到所述结点v的有向边加入E_S,

步骤(3.5),根据步骤(2)及步骤(3.1)至步骤(3.4)的结果得到有效子图$G_S=(V_S,E_S,R)$;

步骤(4),按以下步骤重写所述目标文件,更新时间戳:

步骤(4.1),每一个所述目标文件中的实体都带有一个可见性属性,包括默认default、受保护protected、内部internal、隐藏hidden四种,对每一个不属于有效结点集V_S的实体,若其可见性属性为默认default,则改为隐藏hidden,

步骤(4.2),对每一个所述不属于有效结点集V_S的实体,清除其重定位数据,

步骤(4.3),遍历所述目标文件的符号表,清除不再被任何段引用的符号,

步骤(4.4),根据步骤(4.1)至(4.3)的结果,对所述目标文件进行二进制重写,对于有修改的所述目标文件,其时间戳被自动更新,

步骤(5),按以下步骤链接得到优化化的Android镜像文件:

在所述Makefile中添加用于链接时对段进行垃圾收集的链接选项“--gc-sections”,运行make命令,自动重新链接生成优化后的可执行文件、动态库文件并生成相应的Android镜像文件。

移动终端操作系统基于全系统扩展调用图的自动优化方法

技术领域

[0001] 本发明涉及移动通信领域，特别涉及一种移动终端操作系统基于全系统扩展调用图的自动优化方法。

背景技术

[0002] 在中国，互联网和移动设备的发展相互影响，形成强大的普及浪潮。以智能手机、上网本为代表的移动终端在激烈的竞争中迅猛发展。根据中国互联信息网络中心（CNNIC）第 29 次互联网调查报告称，截止 2011 年 12 月底，中国有 5.13 亿互联网用户，其中手机上网用户 3.56 亿，占网民总体的近 69.4%。

[0003] 以安卓（Android）系统为代表的移动终端开源操作系统的使用比例和影响能力在不断扩大。GARTNER 公司的报告显示，在 2011 年的第四季度，Android 是全球使用最广泛的智能手机操作系统，在全球智能手机中的市场占有率约 50%。

[0004] Android 系统一开始就是用开源、开放的开发方式，受到业界和学术界的广泛关注，自从发布以来，Android 的版本基本上每半年一个台阶地向前发展。其平台的开放性也吸引了大量的第三方开发者进行应用程序开发，Android Market 上的应用程序数量从 2009 年 12 月的 1.6 万迅速增长到 2012 年上半年的 50 万。

[0005] 随着近年移动设备和相关软件的迅速发展，移动设备上的操作系统和应用程序的复杂性、程序体积都比几年前大幅提高，这也就对移动操作系统的优化提出了很多新的需求。

[0006] 在这些快速发展中，有相当数量原来用于桌面和服务器系统的组件仅作少量移植后就用于移动操作系统上，例如 Android 系统自身就使用了一百余个外来的开源软件代码，它们既包括最底层的 Linux 内核，也包括上层的 WebKit 浏览器引擎等。这些组件并非专门针对移动操作系统所编写，移植到移动操作系统以后往往还存在优化空间，但如果对它们一一进行手工优化将耗费大量人力物力资源。因此发明一种能自动进行全系统优化的方法非常有必要。

[0007] Android 系统分为四层，从上到下依次为：

[0008] • 应用程序：使用 Java 语言编写，可通过 Java 本地接口（JNI）调用本地代码。

[0009] • 应用程序框架：使用 Java、C、C++ 编写，向应用程序提供活动管理器、窗口管理器等服务。

[0010] • 系统库和 Android 运行时环境：使用 C/C++ 以及少量汇编编写，为应用程序和应用程序框架提供必要的与系统交互的接口。

[0011] • Linux 内核：操作系统内核，使用为 Android 修改过的 Linux 内核。

[0012] 整个 Android 系统（Android-x86，20120215 版本）包含 2270 万行代码，包括 C、C++、Java 以及其他语言。除了 Linux 内核以外，系统中共有 950 万行本地代码（C、C++ 及少量汇编）。

[0013] Android 系统基于 Linux 内核，但并没有使用其他 Linux 系统常用的 GNU libc（服

务器、桌面 Linux 系统常用）或 uClibc（基于 Linux 的嵌入式系统常用），而是使用了专有的 C 库，称为 Bionic。Bionic 库的部分代码衍生自 BSD，部分专门为 Android 编写，并针对 Android 系统作了优化和精简，删除了 Android 不需要的功能。优化后的 Bionic 库的代码体积显著减小，在 x86 平台上仅有 GNU libc 的 26%，uClibc 的 62%。

[0014] 从 Bionic 的例子可以看出，通用的代码在被用于特定的系统时会有比较大的优化空间。据此可以推测：Android 上的其他组件也会有优化的空间，尤其是那些原本为服务器和桌面系统编写的组件。由于工作量的关系，Android 没有也不可能为系统中的每一个组件都人工进行如此细致的优化。本发明通过一种自动进行全系统优化的方法解决了这个问题。

[0015] 本发明给出移动终端操作系统基于全系统扩展调用图的自动优化方法，从全系统编译后的目标文件中读取信息，基于这些信息构建全系统的扩展调用图，进一步找出扩展调用图中的有效子图，根据有效子图对目标文件进行二进制重写后，链接生成优化后的可执行文件和动态库文件。本发明使用自动的方法进行全系统分析和优化，不影响系统的源代码，优化过程不需人工干预，优化后的系统体积减小，性能有所提升，具有重要的实际意义和应用价值。

发明内容

[0016] 本发明使用程序分析技术，设计出一种移动终端操作系统基于全系统扩展调用图的自动优化方法，方法针对目前广泛采用的移动终端操作系统中的软件优化问题，给出自动优化的方法。以 Android 系统为例，要求考虑除了底层 Linux 内核以外的所有本地代码，对它们进行自动优化，在确保优化后的系统正确性不受影响的前提下，消除无用代码，减小代码体积，提升系统性能。

[0017] 本发明的特征在于，是在计算机中依次按以下步骤实现的：

[0018] 步骤（1）：编译移动终端操作系统的源代码，生成可重定位的目标文件；

[0019] 从源代码服务器中下载源代码，修改相关的 Makefile，向 CFLAGS 添加“-ffunction-sections”参数和“-fdata-sections”参数以使编译器将每个函数和数据对象编译到单独的段中，再运行 make 命令自动编译所述源代码生成可重定位的目标文件，其中函数和数据对象统称为“实体”，所述数据对象既包括源代码中定义的全局和静态变量，也至少包括虚函数表在内的编译器生成的数据对象；

[0020] 步骤（2）：按以下步骤构造有向的全系统扩展调用图：

[0021] 步骤（2.1）：对步骤（1）得到的目标文件，读取以下信息，其中包括 SS、GS、AF 和 AU：

[0022] SS，所述目标文件中所有包含有所述实体的段的名称集合，

[0023] GS，所述目标文件中定义的所有全局符号及其名称，并以关联表的形式存储，以便从全局符号的名称迅速查找所在的段的名称，

[0024] AF，头和尾都属于同一个所述目标文件的有向边形成的集合，

[0025] AU，头属于一个所述目标文件，尾为暂未解析的符号的有向边，其中每一个元素表示为 (u, sym)，其中 u 属于 SS，是所述目标文件中的一个所述实体，sym 为一个外部符号的名，表示被实体 u 引用，但不在同一个所述目标文件中定义的实体，

[0026] 步骤(2.2),根据步骤(2.1)中所得到的信息,合并为一个系统全局的有向图,从而得到所述扩展调用图的 V 和 E,其中 :

[0027] V 为所有所述目标文件 SS 的并集,是一个结点集,其中每一个结点的名称用所对应的一个二元组表示,其中包括目标文件名和段名,

[0028] E 为有向边集,它包括两部分 :$E=E_1 \cup E_2$,其中 E_1 为所有所述目标文件的 AF 的并集,代表了头和尾都属于同一个所述目标文件的有向边所形成的集合,E_2 为头和尾属于不同的所述目标文件的有向边形成的集合,按以下方法得到 :首先,令 $E_2=\varnothing$,再遍历每个所述目标文件的 AU 集合,对其中的每一个元素 (u, sym),查找所有所述目标文件的 GS 集,得到所有名称为 sym 的全局符号的所述实体集合 S[sym],将所有二元组 (u,v) | $_{v \in S[sym]}$ 加入集合 E_2,并将 E_1 与 E_2 取并集得到 E,其中每一个元素是一对结点 (u,v),对应所述系统扩展调用图中一条从 u 到 v 的有向边,当且仅当 u 的重定位数据中有相对 v 的重定位记录时,在所述系统扩展调用图中存在一条从 u 到 v 的边,

[0029] 步骤(2.3),按以下步骤得到入口点集 R :

[0030] 步骤(2.3.1),令 $R=\varnothing$,

[0031] 步骤(2.3.2),把同程序启动代码对应的所述实体作为一个结点加入所述入口点集 R,在 Android 中 _start 符号所对应的实体即为程序启动代码,

[0032] 步骤(2.3.3),把可能通过动态绑定使用的所述实体作为一个结点加入所述入口点集 R,

[0033] 步骤(2.3.4),把位于不以 .text、.data、.rodata、.bss 打头的段中的所述实体作为一个结点加入所述入口点集 R,

[0034] 步骤(2.4),从步骤(2.1)至步骤(2.3)得到的 V、E、R 表示为一个所述的系统调用图 G=(V,E,R) ;

[0035] 步骤(3),按以下步骤从步骤(2)得到的所述系统扩展调用图 G 中得到有效子图 $G_S=(V_S,E_S,R)$,其中 V_S 为有效结点集,E_S 为有效的有向边集,表示为 :

[0036] $$V_s = \bigcup_{u \in R} Desc\ (u)$$

[0037] $E_S = E \cap (V_S \times V_S)$

[0038] 其中 Desc(u) 表示 u 在扩展调用图 G 中包括 u 自身在内的子孙结点的集合,

[0039] 步骤(3.1),令 $V_S = R$, $Q = R$ 表示待访问的结点集,$VisitedV=\varnothing$ 表示已访问的结点集,

[0040] 步骤(3.2),从待访问的结点集 Q 中任取一个结点 u 加入 VisitedV,再把结点 u 在所述系统扩展调用图 G 中的所有直接后继结点加入所述 V_S 中,把述结点 u 的不属于所述已访问过的结点集 VisitedV 的直接后继结点加入待访问的结点集 Q 中,并从所述待访问的结点集 Q 中删除访问过的结点 u,

[0041] 步骤(3.3),重复步骤(3.2),直到所述待访问的结点集 Q 为空,得到有效结点集 V_S,

[0042] 步骤(3.4),令 E_S 为空集,遍历 E 中的所有有向边 (u,v),若结点 u 和结点 v 都属于所述有效结点集 V_S,把从所述结点 u 到所述结点 v 的有向边加入 E_S,

[0043] 步骤(3.5),根据步骤(2)及步骤(3.1)至步骤(3.4)的结果得到有效子图 $G_S=(V_S,E_S,R)$;

[0044] 步骤(4),按以下步骤重写所述目标文件,更新时间戳 :

[0045] 步骤(4.1),每一个所述目标文件中的实体都带有一个可见性属性,包括默认 default、受保护 protected、内部 internal、隐藏 hidden 四种,对每一个不属于有效结点集 V_s 的实体,若其可见性属性为默认 default,则改为隐藏 hidden,

[0046] 步骤(4.2),对每一个所述不属于有效结点集 V_s 的实体,清除其重定位数据,

[0047] 步骤(4.3),遍历所述目标文件的符号表,清除不再被任何段引用的符号,

[0048] 步骤(4.4),根据步骤(4.1)至(4.3)的结果,对所述目标文件进行二进制重写,对于有修改的所述目标文件,其时间戳被自动更新,

[0049] 步骤(5),按以下步骤链接得到优化化的 Android 镜像文件 :

[0050] 在所述 Makefile 中添加用于链接时对段进行垃圾收集的链接选项“--gc-sections”,运行 make 命令,自动重新链接生成优化后的可执行文件、动态库文件并生成相应的 Android 镜像文件。

附图说明

[0051] 图 1 示出根据本发明进行移动终端操作系统基于全系统扩展调用图的自动优化方法的流程示意图。

具体实施方式

[0052] 本发明所述方法采取以下步骤进行 :

[0053] 步骤(1),编译移动终端操作系统的源代码。这一步将生成可重定位的目标文件(.o)。在编译前应先修改相关 Makefile,向 CFLAGS 添加“-ffunction-sections”和“-fdata-sections”参数,使编译器将每个函数和数据对象(数据对象既包括源代码中定义的全局和静态变量,也包括虚函数表等由编译器生成的数据对象,函数和数据对象以下统称为“实体”)编译到单独的段中,以利于后面的分析和优化。

[0054] 步骤(2),构建扩展调用图。扩展调用图可表示为 G=(V,E,R)。其中 :

[0055] V 为结点集,其中每一个结点都与目标文件中一个实体对应,由于在步骤(101)中使用了将每个实体编译到单独的段中的选项,因为每一结点又可对应到目标文件中的一个段。

[0056] E 为有向边集,每个元素是一对结点 (u,v)。(u,v) ∈ E 当且仅当 u 直接使用 v,即在 u 的重定位数据中有相对 v 的重定位记录,此时在扩展调用图中有一条从 u 到 v 的边。

[0057] R 为入口点集,它是 V 的一个子集。如果一个点代表程序的启动代码,或可能在程序运行中通过动态绑定访问,或属于特殊的段,则它是一个入口点。

[0058] 具体构建方法如下 :

[0059] 步骤(2.1)使用 elfutils 或类似的库、工具依次读取每个目标文件。对于每一个目标文件,获取以下信息 :

[0060] SS :该目标文件中所有包含有实体的段的名称的集合 ;

[0061] GS :包含该目标文件中定义的所有全局符号的信息,以关联表的形式存储,以便从全局符号的名称迅速查找它所在的段的名称 ;

[0062] AF :头和尾都属于该目标文件的有向边的集合 ;

[0063] AU：头属于该目标文件，尾为暂未解析的外部符号的有向边的信息，其中每一个元素表示为 (u, sym)，其中 u 属于 SS，是该目标文件中的一个实体，sym 为一个外部符号的名字，表示被 u 引用、但不在同一个目标文件中定义的实体。

[0064] 步骤(2.2)根据步骤(2.1)中所获得的信息，合并为一个全局的有向图。这一步将得到扩展调用图的 V 和 E：

[0065] V 为所有目标文件的 SS 的并集，为了区分来自不同目标文件的段，在 V 中每个结点的名称将用（目标文件名，段名）的二元组表示。

[0066] E 表示包括两部分：$E=E_1 \cup E_2$。其中 E_1 为所有目标文件的 AF 的并集，代表头和尾都属于同一个目标文件的有向边的集合，E_2 为头和尾属于不同目标文件的有向边的集合。对所有目标文件的 AF 取并集即得到 E_1。求 E_2 的方法为：首先令 $E_2=\varnothing$；遍历每个目标文件的 AU 集合，对其中的每一个元素 (u, sym)，查找所有目标文件的 GS 集，获得所有名称为 sym 的全局符号的实体集 S[sym]，将所有二元组 (u, v)（其中 v ∈ S[sym]）加入集合 E_2。最后将 E_1 与 E_2 取并集得到 E。

[0067] 步骤(2.3)得到入口点集 R。具体方法为，首先令 $R=\varnothing$，然后依次将下列结点加入集合 R：

[0068] a. 程序启动代码对应的实体，在 Android 上为 _start 符号所对应的代码；

[0069] b. 可能通过动态绑定使用的实体，包括：

[0070] C/C++ 程序可能使用动态绑定访问的实体：扫描所有目标文件的只读数据段 (rodata)，找到其中所有字符串。对于系统中的所有全局实体(在动态库中的要求对外可见)，如果其名字与其中一个字符串相同，则认为该实体可能通过动态绑定使用，应加入入口点集；

[0071] JNI 入口函数：Java 代码通过 JNI 调用本地代码时，Java 虚拟机动态打开相应的动态库文件，并使用动态绑定的方式使用其中相应的函数。根据 JNI 的标准，JNI_OnLoad、JNI_OnUnload 以及其他 Java_* 函数需标记为入口点(动态注册的 JNI 入口在扩展调用图中已经被注册函数使用，因此无需在此特别标记)；

[0072] 具体的系统中其他可能通过动态绑定使用的实体。

[0073] c. 位于特殊段中的实体，特殊段包括所有名称不以 .text、.data、.rodata、.bss 打头的段。

[0074] 步骤(3)，得到扩展调用图的有效子图。有效子图 $G_s=(V_s, E_s, R)$ 是扩展调用图 $G=(V, E, R)$ 的一个子图。G 与 G_s 的入口结点集 R 相同，V_s 是 V 是子集，E_s 是 E 的子集。V_s 和 E_s 的用公式表示为：

[0075] $$V_s = \bigcup_{u \in R} Desc\ (u)$$

[0076] $$E_s = E \cap (V_s \times V_s)$$

[0077] 其中 Desc(u) 表示 u 在扩展调用图 G 中的所有子孙结点(包括 u 自身)的集合。使用如下步骤计算 V_s：

[0078] 步骤(3.1)首先令 $V_s = R$，以及 Q = R 表示待访问的结点，$VisitedV=\varnothing$ 表示已访问的结点；步骤(3.2)从 Q 中任取一个结点 u，将它从 Q 中删除，加入 VisitedV，再将 u 在 V 中的所有直接后继结点加入 V_s，将 u 的所有不属于 VisitedV 的直接后继结点加入 Q。

[0079] 步骤(3.3)重复步骤(3.2)，直到 Q 为空。

[0080] 步骤(3.4)得到 V_s 后,继续求得 E_s:首先令 E_s 为空集,然后遍历 E 中所有有向边(u,v),如果 u 和 v 都属于 V_s,将从 u 至 v 的有向边加入 E_s。

[0081] 步骤(3.5)根据步骤(3.1)至(3.4)的结果得到有效子图 $G_s=(V_s,E_s,R)$。

[0082] 步骤(4),重写目标文件。依次访问并重写每个目标文件,重写时自动更新了目标文件的时间戳:

[0083] 步骤(4.1)目标文件中的每个实体具有一个可见性属性,可能取值包括默认 default、受保护 protected、内部 internal、隐藏 hidden 四种,对每一个不属于 V_s 的实体,如果其可见性为默认 default,改为隐藏 hidden;

[0084] 步骤(4.2)对每一个不属于 V_s 的实体,清除其重定位数据;

[0085] 步骤(4.3)遍历目标文件的符号表,将不再被任何段引用的符号删除;

[0086] 步骤(4.4)根据步骤(4.1)至步骤(4.3)的结果对目标文件进行重写,对于有修改的目标文件,其时间戳被自动更新。

[0087] 与步骤(2)类似,步骤(4)中对目标文件的操作也可使用 elfutils 等工具。

[0088] 步骤(5),链接得到优化后的系统。在基于 Make 的构建环境中,完成(4)以后,首先修改 Makefile 添加链接选项“--gc-sections”(链接时对段进行垃圾收集),然后直接运行 make 命令。步骤(4)对目标文件进行重写后,目标文件的时间戳仍然比源文件新,因此目标文件不会重新生成,但此时可执行文件和动态库文件的时间戳已经比修改后的目标文件旧,因此将以重写后的目标文件作为输入进行链接,得到优化后的系统。

[0089] 为使本发明的目的、技术方案和优点更加清楚,下面将结合附图,以开源操作系统安卓 Android-x86honeycomb (3.2.3)版本作为实例对本发明的实施方式作进一步地详细描述。

[0090] 1)准备工作。从源代码服务器 www.android-x86.org 下载 2012 年 2 月 15 日的 honeycomb 软件作为源代码。在进行优化前,首先确定未经修改过的系统可以正常编译和运行,编译的配置使用 eeepc-eng,测试环境可以使用 VirtualBox 等虚拟机软件或使用 x86CPU 的笔记本电脑或平板电脑。

[0091] 2)添加适当的编译、链接选项。具体操作为:编辑 build/core/combo/TARGET_linux-x86.mk 文件,添加选项“-ffunction-sections-fdata-sections”到 TARGET_GLOBAL_CFLAGS,添加选项“-Wl,--gc-sections”到 TARGET_GLOBAL_LDFLAGS,添加选项“--strip-unneeded”到 TARGET_STRIP_COMMAND。

[0092] 3)运行 make 命令,将自动编译所有源代码,完成步骤 101。

[0093] 4)使用根据本发明编写的工具,从目标文件中读取信息,分析后并进行二进制重写,完成步骤(2)至(4)。

[0094] 5)再次运行 make 命令,将自动重新链接生成优化后的可执行文件、动态库文件,并生成相应的 Android 镜像文件。

[0095] 6)比较优化前和优化后的系统的代码体积,最终镜像中的所有 ELF 格式文件(Linux 内核除外)的大小比优化前减少 26%。

[0096] 7)在测试环境中运行优化前和优化后的系统,进行对比。使用 Android Monkey 评估系统的正确性,使用相同的伪随机数种子时,最终输出的日志文件除了时间戳以外完全一致。使用 AnTuTu Benchmark 评估系统的性能,整体上约有 1% 的性能提升。

[0097] 以上实施方式仅用于说明本发明，而非对本发明的限制，相关技术领域的普通技术人员，在不脱离本发明的精神和范围的情况下，还可以做出各种变化和变型，因此所有等同的技术方案也属于本发明的范畴，本发明的专利保护范围应由权利要求限定。

[0098] 本发明具有如下优点：

[0099] 1. 应用范围广：上文的说明中虽使用 Android 作为例子，事实上本发明的用户很广泛，凡是在系统上运行的本地代码可以限定在一个集合范围内的系统都可以使用本发明的方法进行优化，很多移动设备上的操作系统都具有此特点。由于本发明的方法在目标文件的层面上进行操作，因此优化方法不依赖于具体的程序设计语言，在 Android 的示例中，本地代码由 C、C++ 和汇编语言编译而来，在使用其他程序设计语言的系统中本发明也可同样有效使用。

[0100] 2. 自动化程度高：根据本发明编写的工具可以无需修改或者只需很少修改就用于不同的系统，对被优化的系统，只需对其构建环境和过程进行很小的改动就可应用本发明的方法，大多数操作都可完全自动完成。

[0101] 3. 使用方便：使用本发明的方法对系统进行优化时，所有修改都在目标代码上进行，无需对源代码作任何修改，可保证不会因为优化而影响系统源代码的可维护性，这对实际的系统开发具有重要意义。

[0102] 4. 优化效果明显：很多研究已经显示真实代码里有很多无用代码，也有不少关于消除无用代码的研究。本发明在整个系统的范围内进行无用代码消除的优化，可以达到更明显的效果。由于无用代码的消除，系统中保留下来的代码的局部性提升，有助于性能的提高。在上文 Android 的示例中，优化后的系统的代码体积比原系统减小 26%，性能整体提升 1%。

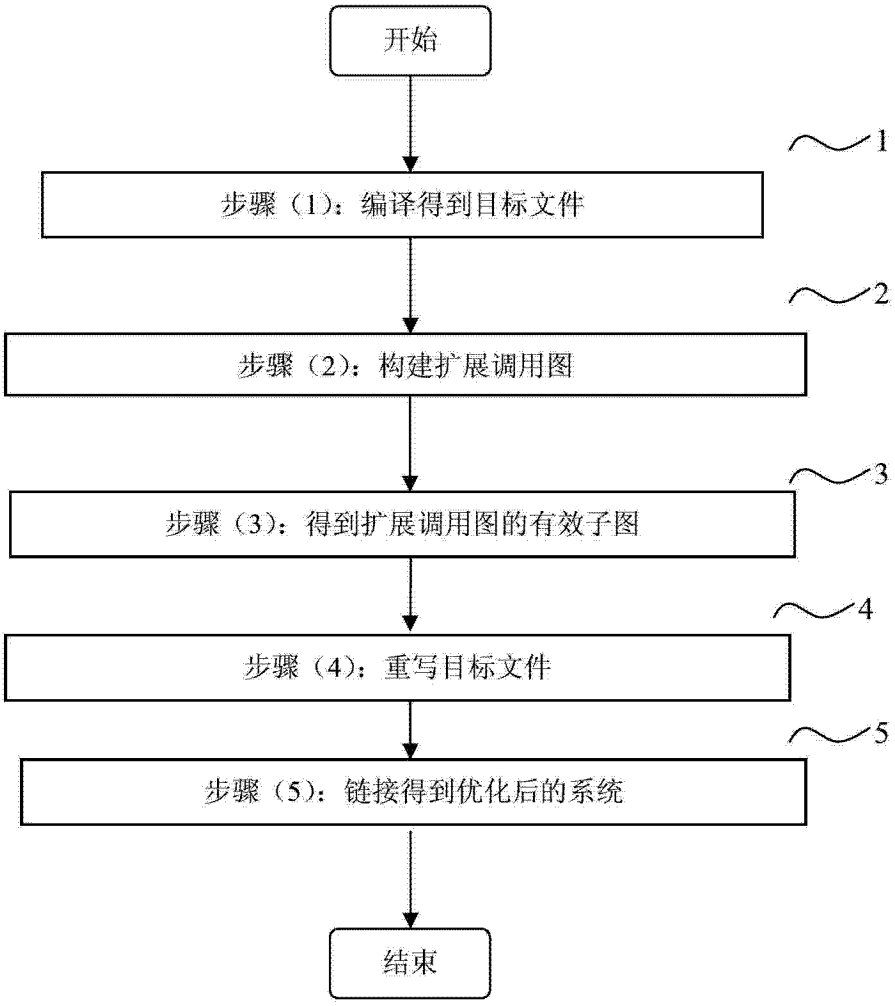

开始
1
步骤（1）：编译得到目标文件
2
步骤（2）：构建扩展调用图
3
步骤（3）：得到扩展调用图的有效子图
4
步骤（4）：重写目标文件
5
步骤（5）：链接得到优化后的系统
结束

图 1