



(12) 发明专利

(10) 授权公告号 CN 102156661 B

(45) 授权公告日 2013. 06. 12

(21) 申请号 201010113646. 2

(22) 申请日 2010. 02. 11

(73) 专利权人 华为技术有限公司

地址 518129 广东省深圳市龙岗区坂田华为
总部办公楼

(72) 发明人 余加强 郑伟

(51) Int. Cl.

G06F 9/48(2006. 01)

G06F 9/45(2006. 01)

(56) 对比文件

CN 101004681 A, 2007. 07. 25,

CN 101482834 A, 2009. 07. 15,

审查员 董泽华

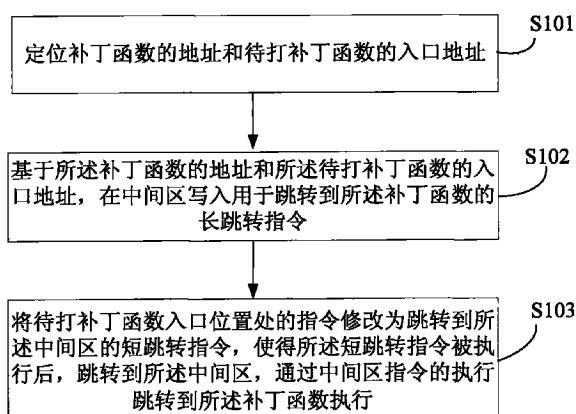
权利要求书2页 说明书11页 附图11页

(54) 发明名称

在线补丁的激活方法、装置及系统

(57) 摘要

本发明实施例公开了一种在线补丁的激活方法、装置及系统，其中，所述方法包括：定位补丁函数的地址和待打补丁函数的入口地址；基于所述补丁函数的地址和所述待打补丁函数的入口地址，在中间区写入用于跳转到所述补丁函数的长跳转指令，其中所述中间区为处于待打补丁函数入口位置前或后，且能放置至少一条长跳转指令的存储空间；将待打补丁函数入口位置处的指令修改为跳转到所述中间区的短跳转指令，使得所述短跳转指令被执行后，跳转到所述中间区，通过中间区中指令的执行跳转到所述补丁函数执行。通过本发明实施例，从而提高应用软件在线补丁激活时的安全性和可靠性。



1. 一种在线补丁激活方法,其特征在于,包括:

定位补丁函数的地址和待打补丁函数的入口地址;

基于所述补丁函数的地址和所述待打补丁函数的入口地址,在中间区写入用于跳转到所述补丁函数的长跳转指令,其中所述中间区为处于待打补丁函数入口位置前或后,且能放置至少一条长跳转指令的存储空间;

将待打补丁函数入口位置处的指令修改为跳转到所述中间区的短跳转指令,使得所述短跳转指令被执行后,跳转到所述中间区,通过所述中间区指令的执行跳转到所述补丁函数执行;

其中,在待打补丁程序编译时,在所述程序的每个函数入口位置前或后预留所述中间区;

根据控制所述中间区的预留及所述中间区的大小的编译选项,编译待打补丁程序的函数生成汇编指令时,在输出函数的汇编函数名之前,输出所述编译选项指定字节数的初始指令以预留中间区。

2. 如权利要求1所述的在线补丁激活方法,其特征在于,被修改为短跳转指令的指令为待打补丁函数入口位置处的首条指令,且长度大于或等于两字节。

3. 如权利要求1所述的在线补丁激活方法,其特征在于,被修改为短跳转指令的指令为待打补丁函数入口位置处的非首条指令,且长度大于或等于两字节,所述方法还包括:在所述中间区写入处于所述被修改指令前的操作指令的反操作指令。

4. 如权利要求3所述的在线补丁激活方法,其特征在于,所述待打补丁函数入口位置处的非首条指令为入栈指令push之后的指令;

所述处于被修改指令前的操作指令的反操作指令为出栈指令pop。

5. 如权利要求1至4任一项所述的在线补丁激活方法,其特征在于,所述在待打补丁程序编译时,在所述程序的每个函数入口位置前或后预留所述中间区的步骤,包括:

将待打补丁程序的源文件编译生成汇编文件后,查找所述汇编文件中表示函数的关键字字符串;

在找到的表示函数的关键字字符串所指示的函数入口位置的前或后,插入指定字节数的初始指令以预留中间区;其中,所述指定字节数表示所述中间区的大小;

对插入有所述初始指令的汇编文件重新编译生成新的汇编文件,并将新的汇编文件编译生成目标文件,由多个目标文件链接生成待打补丁程序的可执行文件。

6. 一种补丁管理装置,其特征在于,所述装置包括:

地址定位单元,用于在待打补丁应用程序的运行过程中,定位与该应用程序关联的补丁函数的地址和待打补丁函数的入口地址;

长跳转指令单元,用于基于所述补丁函数的地址和所述待打补丁函数的入口地址,在中间区写入用于跳转到所述补丁函数的长跳转指令,其中所述中间区为处于待打补丁函数入口位置前或后,且能放置至少一条长跳转指令的存储空间;

短跳转指令单元,用于将待打补丁函数入口位置处的指令修改为跳转到所述中间区的短跳转指令,使得所述短跳转指令被执行后,跳转到所述中间区,通过所述中间区中指令的执行跳转到所述补丁函数执行;

其中,在待打补丁程序编译时,在所述程序的每个函数入口位置前或后预留所述中间

区；

根据控制所述中间区的预留及所述中间区的大小的编译选项，编译待打补丁程序的函数生成汇编指令时，在输出函数的汇编函数名之前，输出所述编译选项指定字节数的初始指令以预留中间区。

7. 如权利要求 6 所述的装置，其特征在于，所述长跳转指令单元具体用于基于所述补丁函数的地址和所述待打补丁函数的入口地址，在中间区写入处于被修改指令前的操作指令的反操作指令和用于跳转到所述补丁函数的长跳转指令，其中所述中间区为处于待打补丁函数入口位置前或后，且能放置至少一条长跳转指令的存储空间；

所述短跳转指令单元具体用于将待打补丁函数入口位置处的长度大于或等于两字节的非首条指令修改为跳转到所述中间区的短跳转指令，使得所述短跳转指令被执行后，跳转到所述中间区，通过所述中间区指令的执行跳转到所述补丁函数执行。

在线补丁的激活方法、装置及系统

技术领域

[0001] 本发明涉及计算机技术领域，具体涉及一种在线补丁的激活方法、装置及系统。

背景技术

[0002] 在线补丁是指程序运行中不重启程序而生效的补丁，广泛应用于各类软件。参见图1，激活在线补丁的基本原理是将要原函数（即待打补丁的函数）的入口处的指令替换为跳转指令，然后通过替换的跳转指令将调用原函数的程序跳转到的补丁函数中执行。随着Linux X86系统在电信领域的广泛应用，同样要求对Linux系统中的应用软件能够在线打补丁，但由于X86系统的指令特点以及Linux的调度方式，使得简单的将被替换函数的入口处指令改为跳转指令的补丁激活方式变得不完全可靠，不能满足电信软件对可靠性的要求。

[0003] 参见图2，在Linux X86系统中，无条件跳转指令占5个字节，激活在线补丁时绝大多数情况下都会覆盖原函数入口处的3条指令，称原函数中这3条指令占用的5个字节的区域为临界区。如果激活在线补丁时直接将原函数入口处的指令替换为跳转指令，则当进程中多个线程时，有可能出现某线程执行到临界区处（如执行到第一条或第二条指令）时刚好发生线程切换的情况，若此时激活在线补丁，该线程切换回来后由于原函数的临界区代码已被新的跳转指令覆盖，程序便会发生异常。

[0004] 现有技术一般使用Pannus补丁技术，具体包括以下步骤：

[0005] (1) 使用函数ptrace将原函数的进程暂停；

[0006] (2) 检查原函数所有线程的EIP(extended instruction pointer, 指令指针寄存器)值是否在临界区；

[0007] (3) 如果没有线程的EIP值在临界区，则在补丁函数入口写入跳转指令，恢复进程的执行；

[0008] (4) 如果有线程的EIP值在临界区，则恢复进程执行一段时间，重新暂停进程进行检查；

[0009] (5) 检查若干次（可自定义，如10次）后，如果还不能激活补丁，则返回激活补丁失败。

[0010] 由于Pannus先暂停原函数进程，再检查所有线程的EIP值是否在临界区，因此，可以在一定程度上避免因直接写入而发生的程序异常。

[0011] 发明人在实现本发明过程中，发现现有技术中：

[0012] 替换函数前只检查当前线程EIP值是否在临界区，一旦有线程是在信号处理函数中，信号的返回地址在临界区内，则在线程处理完信号处理函数后返回时，由于临界区已被跳转指令覆盖，这时将会导致程序出错，因此这种现有技术方案仍然不能保证激活补丁时安全可靠。

发明内容

[0013] 本发明实施例在于提供一种软件在线补丁的激活方法及系统，以提高应用软件在线补丁激活时的安全性和可靠性。

[0014] 本发明实施例是通过以下技术方案实现的：

[0015] 一种在线补丁激活方法，包括如下步骤：

[0016] 定位补丁函数的地址和待打补丁函数的入口地址；

[0017] 基于所述补丁函数的地址和所述待打补丁函数的入口地址，在中间区写入用于跳转到所述补丁函数的长跳转指令，其中所述中间区为处于待打补丁函数入口位置前或后，且能放置至少一条长跳转指令的存储空间；

[0018] 将待打补丁函数入口位置处的指令修改为跳转到所述中间区的短跳转指令，使得所述短跳转指令被执行后，跳转到所述中间区，通过所述中间区指令的执行跳转到所述补丁函数执行。

[0019] 以及，一种通信系统，所述通信系统包括至少一个CPU和内存，所述CPU上运行有操作系统，所述操作系统之上运行有至少一种应用程序，所述应用程序关联有补丁管理线程和至少一个业务线程，其中：

[0020] 所述内存中载入有包含至少一个待打补丁函数的应用程序和包含补丁函数的补丁文件，其中，所述待打补丁函数的入口位置前或后具有能放置至少一条长跳转指令的存储空间；

[0021] 所述补丁管理线程用于在所述应用程序的运行过程中，定位所述补丁函数的地址和所述待打补丁函数的入口地址，并在所述存储空间写入用于跳转到所述补丁函数的长跳转指令，以及将所述待打补丁函数入口位置处的指令修改为用于跳转到所述存储空间的短跳转指令；

[0022] 所述业务线程用于执行到待打补丁函数入口位置处的短跳转指令，跳转到所述存储空间，通过所述存储空间中指令的执行跳转到所述补丁函数执行。

[0023] 以及，一种补丁管理装置，所述装置包括：

[0024] 地址定位单元，用于在待打补丁应用程序的运行过程中，定位与该应用程序关联的补丁函数的地址和待打补丁函数的入口地址；

[0025] 长跳转指令单元，用于基于所述补丁函数的地址和所述待打补丁函数的入口地址，在中间区写入用于跳转到所述补丁函数的长跳转指令，其中所述中间区为处于待打补丁函数入口位置前或后，且能放置至少一条长跳转指令的存储空间；

[0026] 短跳转指令单元，用于将待打补丁函数入口位置处的指令修改为跳转到所述中间区的短跳转指令，使得所述短跳转指令被执行后，跳转到所述中间区，通过所述中间区中指令的执行跳转到所述补丁函数执行。

[0027] 可见，本发明实施例通过在待打补丁程序的函数前或后的存储空间写入跳转到所述补丁函数的长跳转指令，以及将待打补丁函数入口位置处的指令修改为跳转到所述存储空间的短跳转指令，补丁激活时通过该存储空间中指令的执行进行跳转使补丁生效，由于修改函数入口位置处的指令的操作是原子操作，即修改前的指令和修改后的指令的指令长度相同，因而仅修改一条指令即可，无需覆盖临界区其它指令，从而避免了现有技术中对应用软件在线补丁激活时，因系统采用复杂指令集，跳转指令会覆盖函数入口的多条指令，而导致多线程调度机制下所存在的补丁激活安全性和可靠性隐患，因此，本发明实施例方法

可以保证多线程条件下软件在线补丁激活的安全性和可靠性,且不会中断业务。

附图说明

[0028] 为了更清楚地说明本发明实施例或现有技术中的技术方案,下面将对实施例或现有技术描述中所需要使用的附图作简单地介绍,显而易见地,下面描述中的附图仅仅是本发明的一些实施例,对于本领域普通技术人员来讲,在不付出创造性劳动性的前提下,还可以根据这些附图获得其它的附图。

- [0029] 图 1 为现有技术中的补丁激活原理示意图;
- [0030] 图 2 为现有技术中的临界区示意图;
- [0031] 图 3a 为本发明实施例的通信系统的一种结构示意图;
- [0032] 图 3b 为本发明实施例的通信系统的局部逻辑示意图;
- [0033] 图 4 为本发明实施例的在线补丁的制作与管理示意图;
- [0034] 图 5 为本发明实施例的一种软件在线补丁激活方法的流程示意图;
- [0035] 图 6 为本发明实施例的另一种软件在线补丁激活方法的流程示意图;
- [0036] 图 7 为本发明具体实施例中的软件在线补丁激活方法的原理示意图;
- [0037] 图 8 为图 6 中 S201 的一种预留中间区的方法流程示意图;
- [0038] 图 9 为一种传统的应用程序的编译过程的原理示意图;
- [0039] 图 10 为图 6 中 S201 的又一种预留中间区的方法流程示意图;
- [0040] 图 11 为图 10 中 S2022 的一种具体流程示意图;
- [0041] 图 12 为本发明实施例的在汇编文件中插入中间区的前后对比效果示意图;
- [0042] 图 13 为本发明实施例的一种补丁管理装置的结构示意图。

具体实施方式

[0043] 为使本发明实施例的目的、技术方案和优点更加清楚,下面将结合本发明实施例中的附图,对本发明实施例中的技术方案进行清楚、完整地描述,显然,所描述的实施例是本发明一部分实施例,而不是全部的实施例。基于本发明中的实施例,本领域普通技术人员在没有作出创造性劳动前提下所获得的所有其他实施例,都属于本发明保护的范围。

[0044] 请参阅图 3a 和 3b,为本发明实施例的通信系统的结构示意图,本发明实施例通信系统支持多线程环境下的软件在线补丁的激活,需要说明的是,图 3a 中包括三个 CPU(CPU11,12,13),所述 CPU 通过总线 20 访问内存 30,应当理解的是,本发明实施例的通信系统中可以包括一个 CPU,即单 CPU 的通信系统,也可以包括多个 CPU,即多 CPU(多核)的通信系统。即本发明实施例的通信系统,包括至少一个 CPU 和内存,所述 CPU 上运行有操作系统,所述操作系统之上运行有至少一种应用程序(亦称为软件),所述应用程序关联有补丁管理线程和至少一个业务线程,需要说明的是,这里的补丁管理线程可以是待打补丁程序内部的线程,也可以是独立于所有待打补丁程序之外的线程,其中:

[0045] 所述内存中载入有包含至少一个待打补丁函数的应用程序和包含补丁函数的补丁文件,其中,所述待打补丁函数的入口位置前或后具有能放置至少一条长跳转指令的存储空间;这里的存储空间,例如可以是函数入口位置(亦可称为函数起始位置)前或后的 128 字节内,具体可以是函数入口位置前的六个字节。

[0046] 所述补丁管理线程,用于在所述应用程序的运行过程中,定位所述补丁函数的地址和所述待打补丁函数的入口地址,并在所述存储空间写入用于跳转到所述补丁函数的长跳转指令,以及将所述待打补丁函数入口位置处的指令修改为用于跳转到所述存储空间的短跳转指令;所述业务线程,用于执行到待打补丁函数入口位置处的短跳转指令,跳转到所述存储空间,通过所述存储空间中指令的执行跳转到所述补丁函数执行,从而实现在线补丁生效。

[0047] 为了下文描述方便,这里将待打补丁函数入口位置处被修改为短跳转指令的原指令称作指令 A。

[0048] 在一种实现方式下,所述补丁管理线程具体用于在所述应用程序的运行过程中,定位所述补丁函数的地址和所述待打补丁函数的入口地址,在所述存储空间写入用于跳转到所述补丁函数的长跳转指令,或者,在所述存储空间写入处于指令 A 前的操作指令的反操作指令和用于跳转到所述补丁函数的长跳转指令,以及将待打补丁函数入口位置处的长度大于或等于两字节的指令 A 修改为用于跳转到所述存储空间的短跳转指令;

[0049] 具体的,如果所述指令 A 为待打补丁函数入口位置处的首条指令,且长度大于或等于两字节,则在所述存储空间写入用于跳转到所述补丁函数的长跳转指令;或者,如果所述指令 A 为待打补丁函数入口位置处的非首条指令,且长度大于或等于两字节,则在所述存储空间写入处于指令 A 前的操作指令的反操作指令和用于跳转到所述补丁函数的长跳转指令。

[0050] 在一种具体实现方式下,所述补丁管理线程具体用于在所述应用程序的运行过程中,定位所述补丁函数的地址和所述待打补丁函数的入口地址,并在所述存储空间写入出栈指令 pop 和用于跳转到所述补丁函数的长跳转指令,以及将待打补丁函数入口位置的入栈指令 push 之后的 move 指令修改为用于跳转到所述存储空间的短跳转指令。

[0051] 本发明的一种实施例中,本发明实施例的通信系统进一步包括:编译器(图中未示意出),用于在对待打补丁程序进行编译时,在所述待打补丁程序的每个函数入口位置前或后预留能放置至少一条长跳转指令的存储空间,这里的处于待打补丁函数的入口位置前或后,且能放置至少一条长跳转指令的存储空间可以被称作中间区。

[0052] 在一种具体实现方式下,所述编译器具体用于:根据控制所述存储空间的预留及所述存储空间的大小的编译选项,编译待打补丁的程序相关的函数生成汇编指令时,在输出函数的汇编函数名之前,输出所述编译选项指定字节数的初始指令以预留能放置至少一条长跳转指令的存储空间(中间区)。

[0053] 在另一种具体实现方式下,所述编译器具体用于:将待打补丁程序的源文件编译生成汇编文件后,查找所述汇编文件中表示函数的关键字字符串,并在找到的表示函数的关键字字符串所指示的函数入口位置的前或后,插入指定字节数的初始指令以预留能放置至少一条长跳转指令的存储空间(中间区);对插入有前述初始指令的汇编文件重新编译生成新的汇编文件,并将新的汇编文件生成目标文件,由多个目标文件链接生成待打补丁程序的可执行文件;其中,所述指定字节数表示所述存储空间(中间区)的大小。

[0054] 本发明的另一种实施例下,本发明实施例的通信系统中,所述操作系统之上进一步运行有编译程序,所述编译程序用于在对待打补丁程序进行编译时,在所述待打补丁程序的每个函数入口位置前或后预留能放置至少一条长跳转指令的存储空间。所述编译程序

所涉及的具体工作过程,可以参考前述编译器揭露的相关内容,在此不再赘述。

[0055] 需要说明的是,这里的补丁管理线程和至少一个业务线程可以属于同一个进程,即补丁操作能在单进程内完成;这样,单进程的所有线程能共享访问用户态存储区(即进程空间),在单进程中,补丁管理线程将长跳转指令写入待打补丁函数入口位置前或后且能放置至少一条长跳转指令的存储空间(中间区),以及待打补丁函数入口位置处的指令修改为跳转到所述存储空间的短跳转指令。

[0056] 另一种实现下,这里的补丁管理线程和所述至少一个业务线程也可以分别属于不同的进程,例如:补丁管理线程属于进程A,所述至少一个业务线程属于进程B。

[0057] 需要说明的是,任务是代码运行的一个映象,从系统的角度看,任务是竞争系统资源的最小运行单元。任务可以使用或等待CPU、I/O设备及内存空间等系统资源,并独立于其它任务,与它们一起并发运行。在Linux和Win32系统下,任务对应线程(thread)的概念。

[0058] 应当理解的是,本发明实施例通信系统具体可以是Linux X86系统,或者,Linux X64系统,也可以是Solaris、aix等类Unix系统等等。例如,在Linux X86系统下,包括至少一个x86架构的CPU,CPU上运行有Linux操作系统,所述Linux操作系统之上运行有至少一种应用程序(亦称为软件),所述应用程序关联有补丁管理线程和至少一个业务线程(具体功能同上,故不再赘述)。

[0059] 可见,本发明实施例通过在待打补丁程序的函数入口位置前或后的存储空间写入跳转到所述补丁函数的长跳转指令,以及将待打补丁函数入口位置处的指令修改为跳转到所述存储空间(中间区)的短跳转指令,补丁激活时通过该中间区中指令的执行进行跳转使补丁生效,由于修改函数入口位置处的指令的操作是原子操作,即修改前的指令和修改后的指令的指令长度相同,因而仅修改一条指令即可,无需覆盖临界区其它指令,从而避免了现有技术中对应用软件在线补丁激活时,因系统采用复杂指令集,跳转指令会覆盖函数入口的多条指令,而导致多线程调度机制下所存在的补丁激活安全性和可靠性隐患(即某线程执行到临界区(即待打补丁的函数入口处的指令区域)处时刚好发生线程切换的情况,若此时激活在线补丁,该线程切换回来后由于原函数的临界区已被新的跳转指令覆盖,程序便会发生异常;或者,有线程处理完信号处理函数后返回(信号的返回地址在临界区内)时,由于临界区已被跳转指令覆盖,导致程序出错),因此,本发明实施例方法可以保证多线程条件下软件在线补丁激活的安全性和可靠性,且不会中断业务。

[0060] 请参阅图4,为本发明实施例在线补丁的制作与管理示意图,如图4所示,补丁制作过程为:在线补丁在后台使用补丁工具制作完毕后,补丁源代码被编译成目标文件,并通过补丁制作工具将待打补丁程序的符号文件(绝对或相对定位形式)和补丁程序的目标文件制作生成在线补丁文件。

[0061] 补丁管理过程为:读取上述在线补丁文件(亦称为热补丁文件)到内存,加载补丁到补丁区,并激活生效。

[0062] 下面结合附图来详细描述本发明实施例:

[0063] 请参阅图5,为本发明实施例的一种在线补丁激活方法的流程示意图,该方法可以应用于包括至少一个CPU和内存的通信系统,所述CPU上运行有操作系统,所述操作系统之上运行有至少一种应用程序(亦称为软件),所述应用程序关联有补丁管理线程和至少一

个业务线程,具体的,该方法的执行主体可以是待打补丁程序关联的补丁管理线程,其中该方法可以包括:

[0064] S101、定位补丁函数的地址和待打补丁函数的入口地址;

[0065] 具体的,在待打补丁程序的运行过程中,定位所述补丁函数的地址和所述待打补丁函数的入口地址,其中,具体的是,从内存中的数组中读取补丁函数地址信息;

[0066] 需要说明的是,如图4所示,如果补丁文件中的补丁函数地址信息是绝对地址,在加载补丁的过程中,直接将补丁文件中包含的补丁函数地址信息存入内存中分配好的数组中;

[0067] 如果补丁文件中的补丁函数地址信息是相对地址(非绝对地址),在加载补丁的过程中,根据补丁函数的相对地址计算得到补丁函数的绝对地址,并将计算结果存入内存中分配好的数组中。

[0068] S102、基于所述补丁函数的地址和所述待打补丁函数的入口地址,在中间区写入用于跳转到所述补丁函数的长跳转指令,其中所述中间区为处于待打补丁函数入口位置前或后,且能放置至少一条长跳转指令的存储空间;

[0069] 这里的存储空间(中间区)例如可以是待打补丁函数入口位置(亦可称为函数起始位置)前或后的128字节内,具体可以是函数入口位置前的六个字节。

[0070] S103、将待打补丁函数入口位置处的指令修改为跳转到所述中间区的短跳转指令,使得所述短跳转指令被执行后,跳转到所述中间区,通过中间区中指令的执行跳转到补丁函数执行,从而实现在线补丁生效。

[0071] 为了下文描述方便,这里将待打补丁函数入口位置处被修改为短跳转指令的原指令称作指令A。

[0072] 在一种实现方式下,指令A为待打补丁函数入口位置处的首条指令,且长度大于或等于两字节。

[0073] 在另一种实现方式下,指令A为待打补丁函数入口位置处的非首条指令,且长度大于或等于两字节,则S102中所述在中间区写入用于跳转到所述补丁函数的长跳转指令的步骤具体可以为:在所述中间区写入处于指令A前的操作指令的反操作指令和用于跳转到所述补丁函数的长跳转指令。

[0074] 本发明的一种具体实施例中,指令A为将待打补丁函数入口位置处的入栈指令push之后的指令(即第二条指令),所述处于指令A前的操作指令push的反操作指令为出栈指令pop;相应的,在所述中间区写入出栈pop指令和用于跳转到所述补丁函数的长跳转指令;将待打补丁函数入口位置处的入栈指令push之后的指令(即第二条指令)修改为跳转到所述中间区的短跳转指令。

[0075] 其中,可以在待打补丁的程序编译时在所述程序的每个函数入口位置前或后预留中间区,在一种实现方式下,通过如下方法在所有待打补丁函数入口位置前或后预留中间区,具体包括如下步骤:

[0076] 根据控制所述中间区的预留及所述中间区的大小的编译选项,编译待打补丁程序的函数生成汇编指令时,在输出函数的汇编函数名之前,输出所述编译选项指定字节数的初始指令以预留中间区。应当理解的是,这里的初始指令是用于预先占用存储空间的,相应的,在中间区写入用于跳转到所述补丁函数的长跳转指令即将存储空间中原有的初始指令

修改为长跳转指令（和反操作指令）。

[0077] 在另一种实现方式下，通过如下方法在所有待打补丁函数入口位置前或后预留中间区，具体包括如下步骤：

[0078] 将待打补丁程序的源文件编译生成汇编文件后，查找所述汇编文件中表示函数的关键字字符串；

[0079] 在找到的表示函数的关键字字符串所指示的函数入口位置的前或后，插入指定字节数的初始指令以预留中间区；其中，所述指定字节数表示所述中间区的大小；

[0080] 对插入有所述初始指令的汇编文件重新编译生成新的汇编文件，并将新的汇编文件编译生成目标文件，由多个目标文件链接生成待打补丁程序的可执行文件。

[0081] 可见，本发明实施例通过在待打补丁程序的函数入口位置前或后的存储空间写入跳转到所述补丁函数的长跳转指令，以及将待打补丁函数入口位置处的指令修改为跳转到所述存储空间（中间区）的短跳转指令，补丁激活时通过该中间区中指令的执行进行跳转使补丁生效，由于修改函数入口位置处的指令的操作是原子操作，即修改前的指令和修改后的指令的指令长度相同，因而仅修改一条指令即可，无需覆盖临界区其它指令，从而避免了现有技术中对应用软件在线补丁激活时，因系统采用复杂指令集，跳转指令会覆盖函数入口的多条指令，而导致多线程调度机制下所存在的补丁激活安全性和可靠性隐患（即某线程执行到临界区（即待打补丁的函数入口处的指令区域）处时刚好发生线程切换的情况，若此时激活在线补丁，该线程切换回来后由于原函数的临界区已被新的跳转指令覆盖，程序便会发生异常；或者，有线程处理完信号处理函数后返回（信号的返回地址在临界区内）时，由于临界区已被跳转指令覆盖，导致程序出错），因此，本发明实施例方法可以保证多线程条件下软件在线补丁激活的安全性和可靠性，且不会中断业务。

[0082] 请参阅图6，为本发明实施例的另一种在线补丁激活方法的流程示意图，该方法可以应用于包括至少一个CPU和内存的通信系统，所述CPU上运行有操作系统，所述操作系统之上运行有至少一种应用程序（亦称为软件），所述应用程序关联有补丁管理线程和至少一个业务线程，具体的，该方法的执行主体可以是待打补丁程序关联的补丁管理线程，其中该方法可以包括：

[0083] S201、在一应用程序编译时，在所述应用程序的每个函数入口位置前或后预留中间区；

[0084] 较优的，在函数入口位置前预留中间区，以保证使用短跳转 short jmp 指令能够跳转到中间区。如图7所示，在本发明具体实施例中，所述中间区处于待打补丁函数入口位置前，且占用六个字节（相对跳转指令）。如果使用绝对跳转指令或者是应用于x86-64位系统，则预留中间区的地址空间应相应扩大。

[0085] S202、当该应用程序需要在线打补丁时，在该应用程序的运行过程中加载包括补丁函数的补丁文件到所述内存中，并将补丁函数地址信息存入内存；

[0086] 具体是，将补丁文件中包含的补丁函数地址信息存入内存中的数组或经计算后存入内存中的数组；

[0087] S203、当触发补丁激活的事件发生时，基于所述待打补丁函数的入口地址和从内存中读取的补丁函数地址信息，在中间区写入待打补丁函数入口位置处的push指令的反操作指令pop指令（即出栈指令，占用1个字节）和用于跳转到相应补丁函数的长跳转指

令 long jmp ;

[0088] 如图 7 所示,在本发明具体实施例中,这里的长跳转指令 long jmp 可以是 5 字节的相对跳转指令,或者占用更多字节的绝对跳转指令。

[0089] 如图 7 所示,在本发明具体实施例中,写入中间区的指令为一个 pop 指令和一个 long jmp 指令。在另一种实现方式下,写入中间区的指令也可以是一个 push 指令,两个 pop 指令和一个 long jmp 指令;同理,在又一种实现方式下,写入中间区的指令也可以是两个 push 指令,三个 pop 指令和一个 long jmp 指令;应当理解的是,中间区中包含的 pop 指令的个数与临界区中包含的 push 指令和中间区中包含的 push 指令的个数之和相等,换言之,即总的入栈指令和总的出栈指令个数相当。

[0090] S204、将待打补丁函数入口位置处的 push 指令(即入栈指令)之后的指令修改为用于跳转到该中间区的短跳转指令 short jump 指令(其占用 2 字节),使得所述 short jump 指令被执行后跳转到所述中间区,通过中间区中指令的执行跳转到补丁函数执行,从而实现在线补丁的激活 / 生效。

[0091] 需要说明的是,在又一种实现方式下,如果待打补丁函数的入口位置处待修改为短跳转指令的原指令为待打补丁函数的入口位置处的首条指令且指令长度大于或等于 2 个字节,则 S203 中可以在中间区写入用于跳转到补丁函数的长跳转指令 long jmp 即可。

[0092] 如图 7 所示,在本发明具体实施例中,将待打补丁函数入口位置处的 push 指令之后的 move 指令修改为用于跳转到所述中间区的短跳转指令 short jump 指令,修改前的 move 指令与修改后的 short jmp 指令均占用 2 字节,指令长度相同,这里的 move 指令即待打补丁函数入口位置处的第二条指令。由于修改待打补丁函数入口位置处的第二条指令的操作是原子操作,本发明实施例方法可以保证在线补丁生效的安全性和可靠性。

[0093] 可见,本发明实施例通过在待打补丁程序的函数入口位置前或后预留中间区,并在所述预留的中间区写入跳转到所述补丁函数的长跳转指令,以及将待打补丁函数入口位置的 push 指令之后的指令即第二条指令修改为跳转到所述中间区的短跳转指令,补丁激活时通过该中间区进行跳转使补丁生效,由于修改函数入口位置的第二条指令的操作是原子操作,即修改前的指令和修改后的指令的指令长度相同,因而仅修改一条指令即可,无需覆盖临界区其它指令,从而避免了现有技术中对应用软件在线补丁激活时,因系统采用复杂指令集,跳转指令会覆盖函数入口的多条指令,而导致多线程调度机制下所存在的补丁激活安全性和可靠性隐患(即某线程执行到临界区处时刚好发生线程切换的情况,若此时激活在线补丁,该线程切换回来后由于原函数的临界区已被新的跳转指令覆盖,程序便会发生异常;或者,有线程处理完信号处理函数后返回(信号的返回地址在临界区内)时,由于临界区已被跳转指令覆盖,导致程序出错),因此,本发明实施例方法可以保证多线程条件下软件在线补丁激活的安全性和可靠性,且不会中断业务。

[0094] 请参阅图 8,为本发明实施例的一种预留中间区的方法的流程示意图,支持在被打补丁的程序编译时在该程序的每个函数入口位置前或后预留中间区,其中该方法应用于使用开源编译器 gcc 的系统中,本发明实施例预留中间区的操作可以通过对编译器代码的修改实现,其中包括如下步骤:

[0095] S2011、在用于待打补丁程序编译的编译器的代码中添加编译选项,所述编译选项用于控制中间区的预留及中间区的大小;

[0096] S2012、根据所述编译选项,编译待打补丁程序的函数生成汇编指令时,在输出函数的汇编函数名之前,输出所述编译选项指定字节数的初始指令以预留中间区。

[0097] 这里的初始指令,例如可以是全空 null 的空指令,或全 0 的指令。

[0098] 相应的,图 6 中 S203 中在中间区写入 pop 指令和长跳转指令 long jmp 的步骤具体为:将初始指令修改为 pop 指令和长跳转指令 long jmp。

[0099] 需要说明的是,在开源编译器 gcc 的系统的一般版本中,可以在 Varasm.c 文件中的 assemble_start_function 中输出函数 prefix 到汇编文件的代码之前,增加向汇编文件输出所述编译选项指定字节数的初始指令的操作。

[0100] 请参阅图 9,为一种传统的应用程序的编译过程的原理示意图,如图 9 所示,传统的编译过程包括:通过编译器将源文件 (.c 文件) 编译成目标文件 (.o 文件,即二进制文件),再将多个目标文件链接成可执行文件。

[0101] 请参阅图 10,为本发明实施例的又一种预留中间区的方法的流程示意图,支持在被打补丁的程序编译时在所有函数入口位置前或后预留中间区,本发明实施例预留中间区的操作可以通过修改编译过程中产生的汇编文件来实现中间区的预留。该方法不仅适用于使用开源编译器 gcc 编译的应用程序,而且适用于使用不开源的其它编译器编译的应用程序,其中该方法具体介绍在函数入口位置前预留中间区,包括如下步骤:

[0102] S2021、将待打补丁程序的源文件 (.c 文件) 编译成汇编文件 (.s 文件);由于为现有技术,故这里不再赘述。

[0103] S2022、在所有函数入口位置前预留中间区:查找所述编译生成的汇编文件中表示函数的关键字字符串,并在找到的每个表示函数的关键字字符串所指示的函数入口位置前,插入指定字节数的初始指令以预留中间区;

[0104] 这里的初始指令,例如可以是全空 null 的空指令,或全 0 的指令。这里以指定字节数来表示预留的中间区的大小。

[0105] S2023、对插入有所述初始指令的汇编文件重新编译生成新的汇编文件;由于为现有技术,故这里不再赘述。

[0106] S2024、将所述新的汇编文件编译生成目标文件;由于为现有技术,故这里不再赘述。

[0107] S2025、将多个目标文件链接生成待打补丁程序的可执行文件,由于为现有技术,故这里不再赘述。需要说明的是,这里生成的可执行文件即所有函数入口位置前已预留中间区的待打补丁程序。

[0108] 其中,如图 11 所示,S2022 中在函数入口位置前预留中间区的过程具体包括:

[0109] S2022a、设置中间区的大小和待插入的初始指令;这里中间区的大小,以指定字节数来表示。

[0110] 如图 12a、12b 所示,本发明一种具体实施例中,这里的初始指令具体为全 0 的指令 0×90。指定字节数具体为 6 个字节 byte,其中图 12a 为在找到的函数入口位置前插入 6 个字节数的全 0 指令的操作前的示意图;图 12b 为在找到的函数入口位置前插入 6 个字节数的全 0 指令的操作后的示意图。

[0111] S2022b、在汇编文件中查找表示函数的关键字字符串,如果找到关键字字符串,则表示找到函数入口位置,转到 S2022c;如果没有找到,则转到 S2022d,结束对汇编文件的操

作；

[0112] 如图 12 所示，本发明一种具体实施例中，这里的表示函数的关键字字符串为 @function，如图 12 所示，查找到两处 @function。

[0113] S2022c、在找到的函数入口位置前插入指定字节数的初始指令，返回执行 S2022b，继续查找下一处表示函数的关键字字符串，直到找到所有函数并在每个函数入口位置前插入指定字节数的初始指令。汇编文件中插入中间区的效果如图 12 所示，图 12 中对比表示了在找到的函数入口位置前插入 6 个字节数的全 0 指令的操作前后的示意图。

[0114] 请参阅图 13，为本发明实施例的一种补丁管理装置的结构示意图，需要说明的是，本发明实施例的补丁管理装置可以理解成前述的补丁管理线程，而本发明实施例涉及的补丁管理线程的表现形式可以是独立于各种应用程序之外的一种补丁管理程序，或者是待打补丁的应用程序内部的线程，如图 13 所示，本实施例的补丁管理装置包括：

[0115] 地址定位单元 301，用于在待打补丁应用程序的运行过程中，定位与该应用程序关联的补丁函数的地址和待打补丁函数的入口地址；

[0116] 长跳转指令单元 302，用于基于所述补丁函数的地址和所述待打补丁函数的入口地址，在中间区写入用于跳转到所述补丁函数的长跳转指令，其中所述中间区为处于待打补丁函数入口位置前或后，且能放置至少一条长跳转指令的存储空间；这里的存储空间（中间区）例如可以是待打补丁函数入口位置（亦可称为函数起始位置）前或后的 128 字节内，具体可以是函数入口位置前的六个字节。

[0117] 短跳转指令单元 303，用于将待打补丁函数入口位置处的指令修改为跳转到所述中间区的短跳转指令，使得所述短跳转指令被执行后，跳转到所述中间区，通过所述中间区中指令的执行跳转到所述补丁函数执行。

[0118] 为了下文描述方便，这里将待打补丁函数入口位置处被修改为短跳转指令的原指令称作指令 A。

[0119] 在一种实现方式下，指令 A 为待打补丁函数入口位置处的首条指令，且长度大于或等于两字节。

[0120] 在另一种实现方式下，指令 A 为待打补丁函数入口位置处的非首条指令，且长度大于或等于两字节，相应的，长跳转指令单元 302 具体用于基于所述补丁函数的地址和所述待打补丁函数的入口地址，在中间区写入处于待修改指令（即指令 A）前的操作指令的反操作指令和用于跳转到所述补丁函数的长跳转指令，其中所述中间区为处于待打补丁函数入口位置前或后，且能放置至少一条长跳转指令的存储空间；

[0121] 相应的，短跳转指令单元 303 具体用于将待打补丁函数入口位置处的长度大于或等于两字节的非首条指令修改为跳转到所述中间区的短跳转指令，使得所述短跳转指令被执行后，跳转到所述中间区，通过所述中间区指令的执行跳转到所述补丁函数执行。

[0122] 具体地，本实施例中，所有单元所涉及的具体工作过程，可以参考上述在线补丁激活方法所涉及的相关实施例揭露的相关内容，在此不再赘述。

[0123] 综上所述，本发明前述实施例通过在待打补丁程序的函数入口位置前或后预留中间区，并在所述预留的中间区写入跳转到所述补丁函数的长跳转指令，以及将待打补丁函数入口位置处的指令修改为跳转到所述中间区的短跳转指令，补丁激活时通过该中间区进行跳转使补丁生效，由于修改函数入口位置处的指令的操作是原子操作，即修改前的指令

和修改后的指令的指令长度相同,因而仅修改一条指令即可,无需覆盖临界区其它指令,从而避免了现有技术中对应用软件在线补丁激活时,因系统采用复杂指令集,跳转指令会覆盖函数入口的多条指令,而导致多线程调度机制下所存在的补丁激活安全性和可靠性隐患(即某线程执行到临界区处时刚好发生线程切换的情况,若此时激活在线补丁,该线程切换回来后由于原函数的临界区已被新的跳转指令覆盖,程序便会发生异常;或者,有线程处理完信号处理函数后返回(信号的返回地址在临界区内)时,由于临界区已被跳转指令覆盖,导致程序出错),因此,本发明实施例方法可以保证多线程条件下,软件在线补丁激活的安全性和可靠性,且不会中断业务。

[0124] 本领域普通技术人员可以理解实现上述实施例方法中的全部或部分流程,是可以通过计算机程序来指令相关的硬件来完成,所述的程序可存储于一计算机可读取存储介质中,该程序在执行时,可包括如上述各方法的实施例的流程。其中,所述的存储介质可为磁碟、光盘、只读存储记忆体(Read-OnlyMemory, ROM)或随机存储记忆体(Random Access Memory, RAM)等。

[0125] 以上举较佳实施例,对本发明的目的、技术方案和优点进行了进一步详细说明,所应理解的是,以上所述仅为本发明的较佳实施例而已,并不用以限制本发明,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。

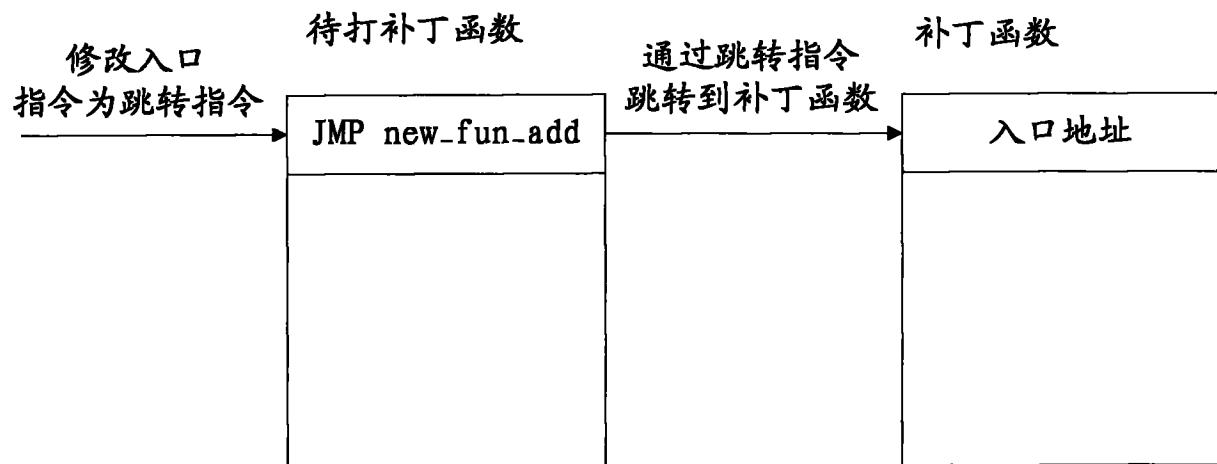


图 1

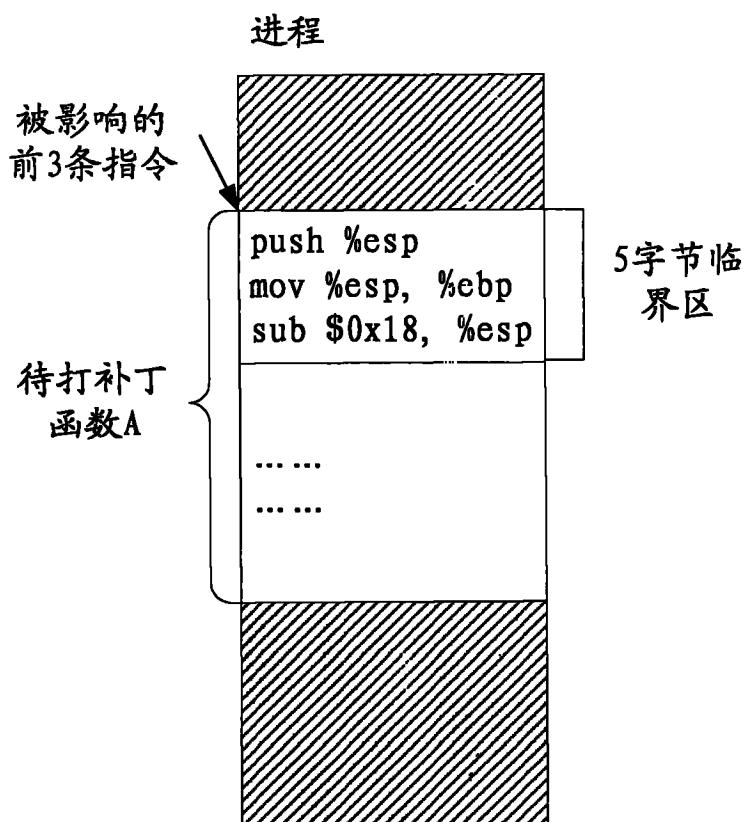


图 2

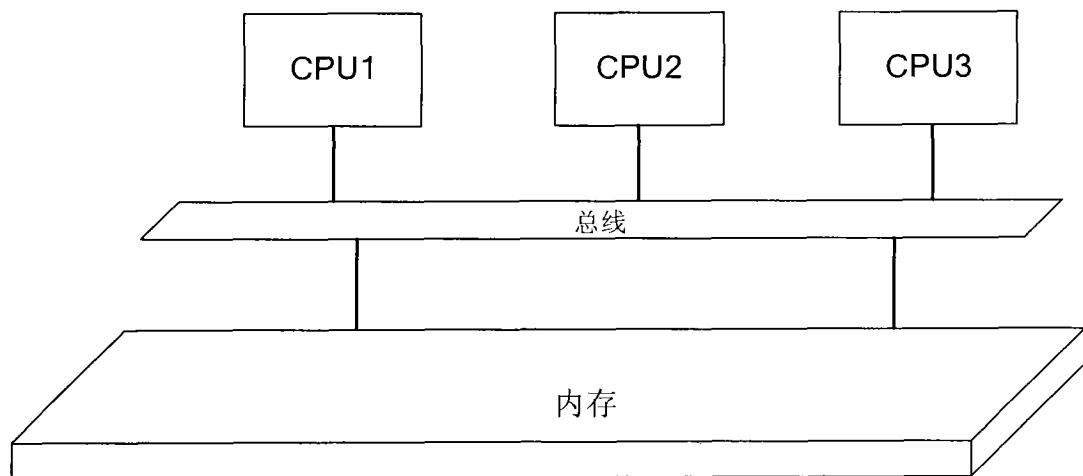


图 3a

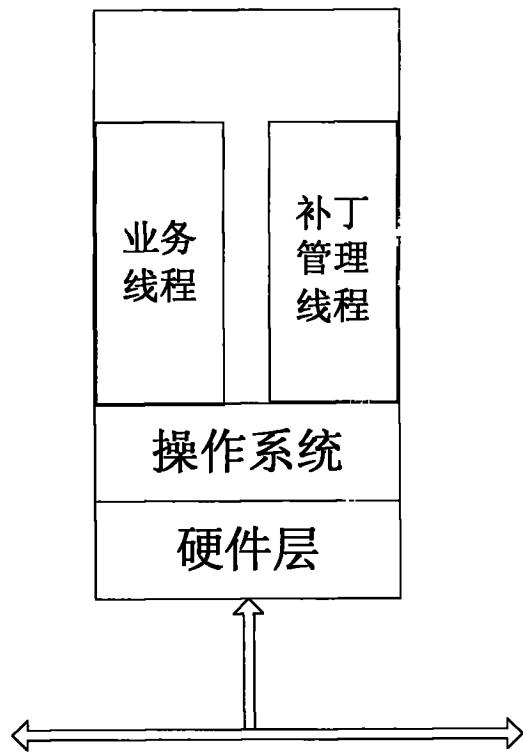


图 3b

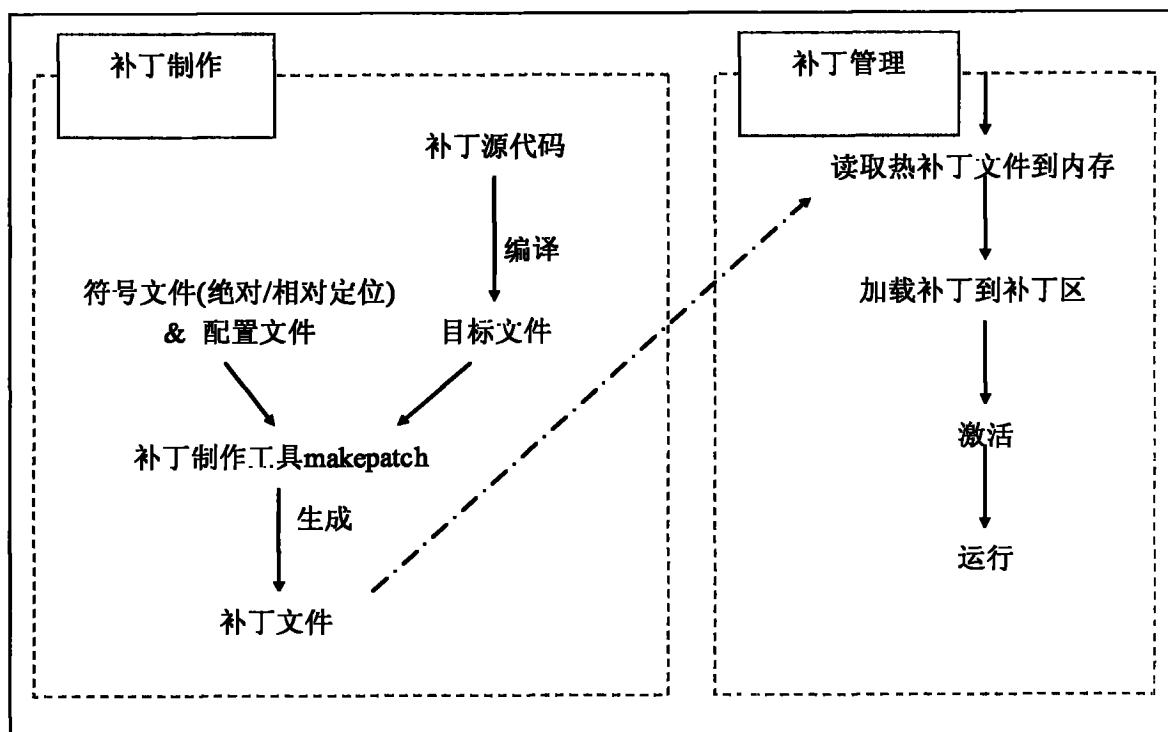


图 4

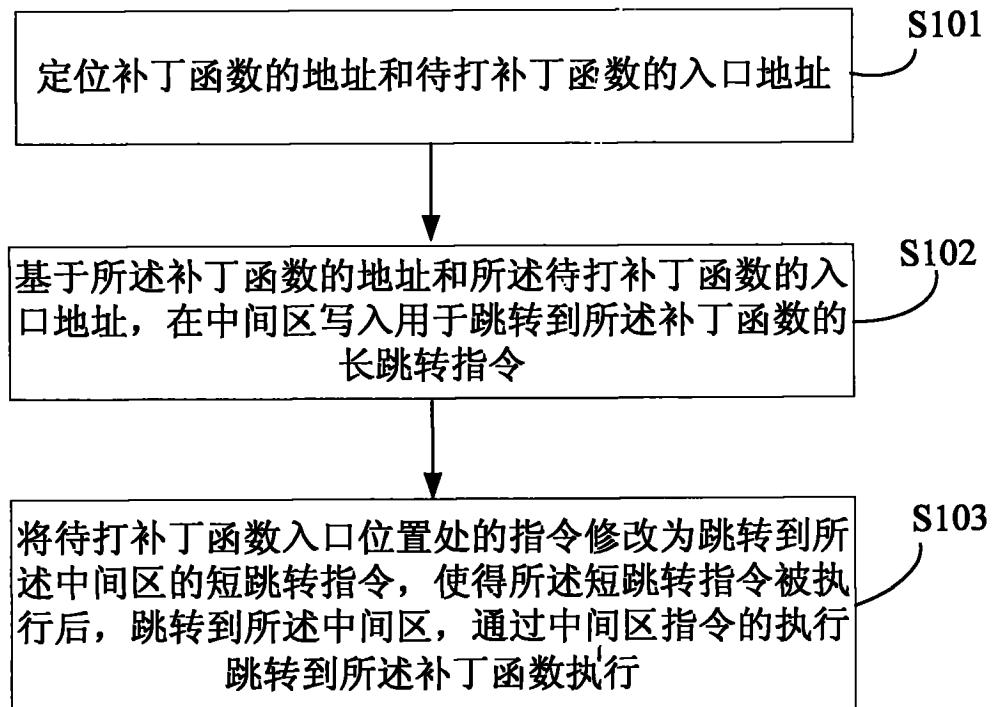


图 5

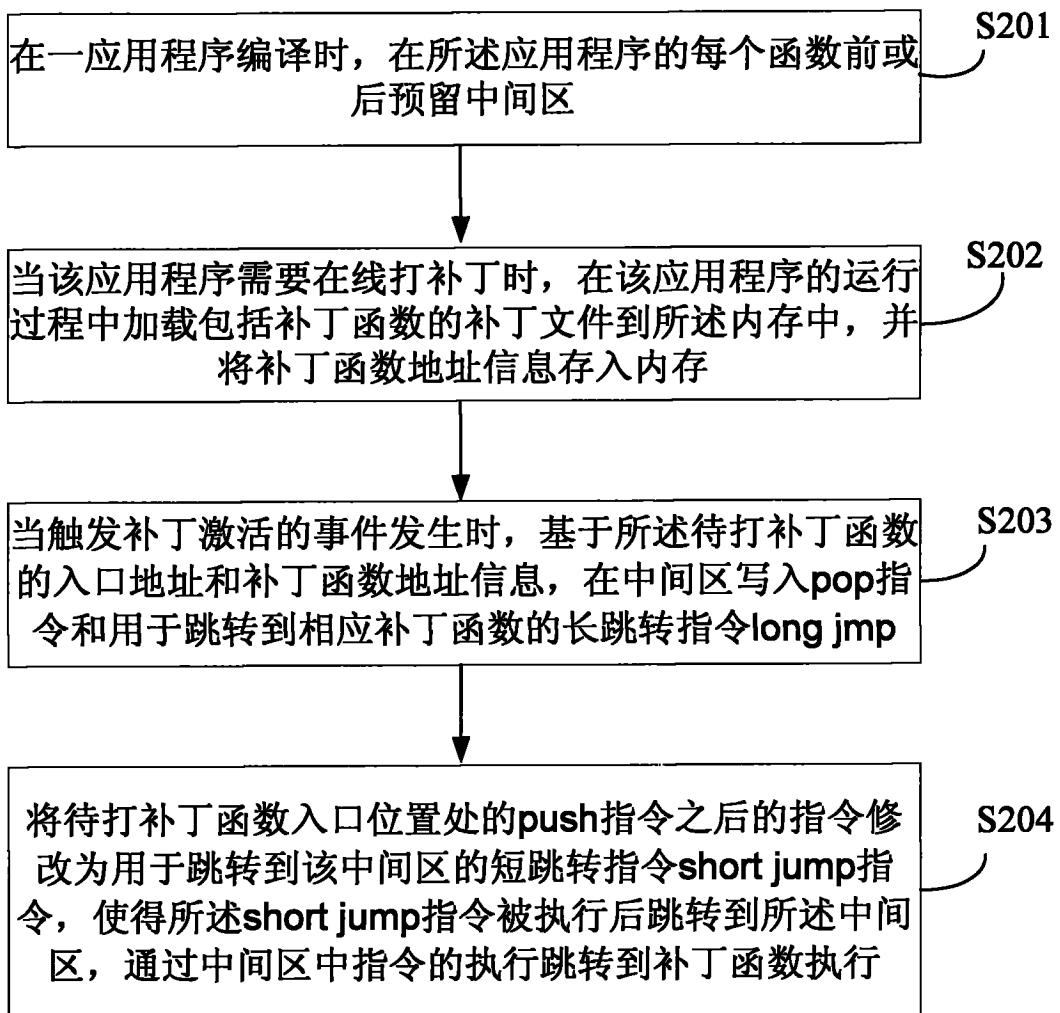


图 6

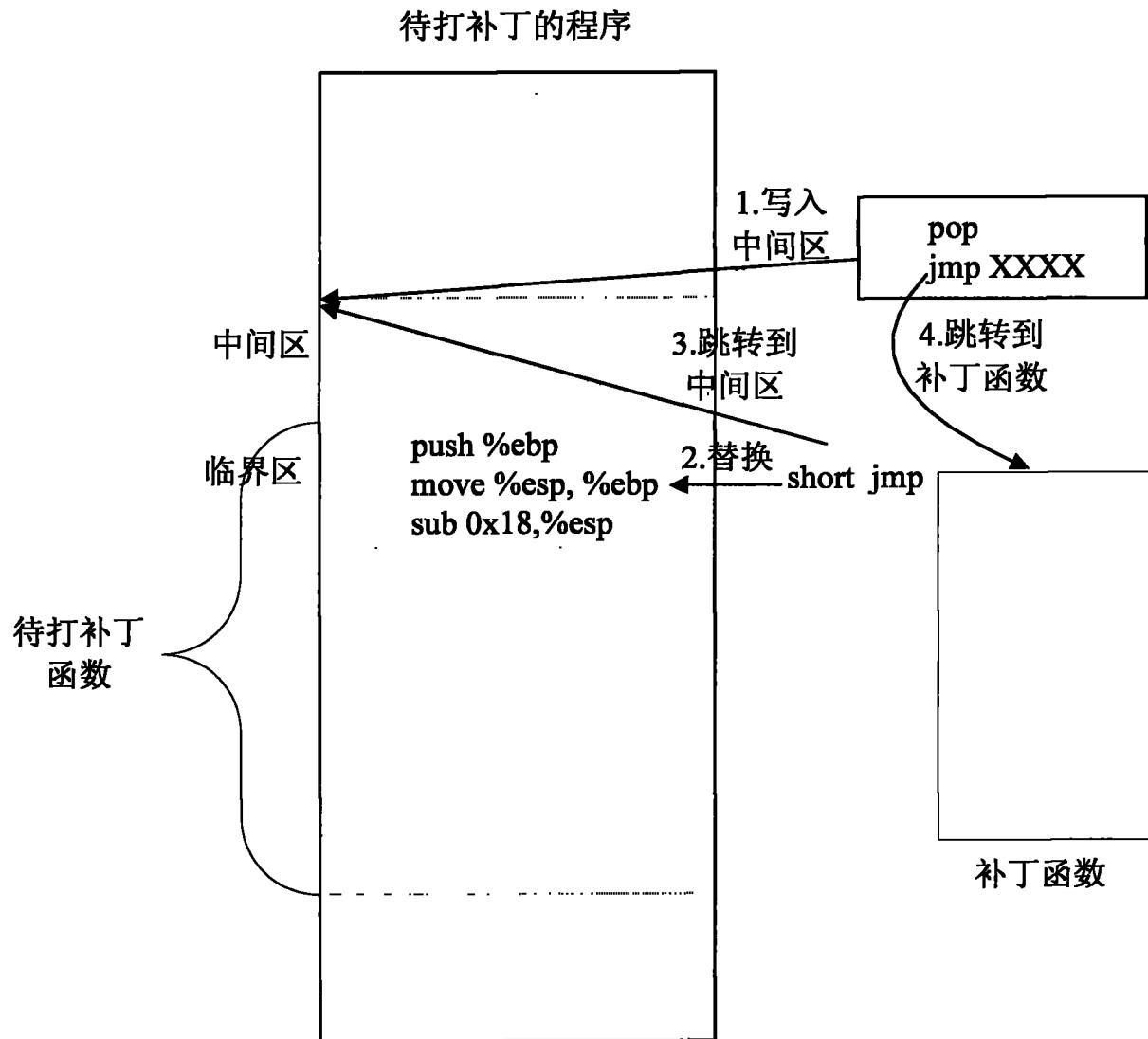


图 7

在用于待打补丁程序编译的编译器代码中添加编译选项，
所述编译选项用于控制中间区的预留及中间区的大小

S2011

根据所述编译选项，编译待打补丁程序的函数生成汇编指令时，在输出函数prefix之前，输出所述编译选项指定字节数的初始指令以预留中间区

S2012

图 8

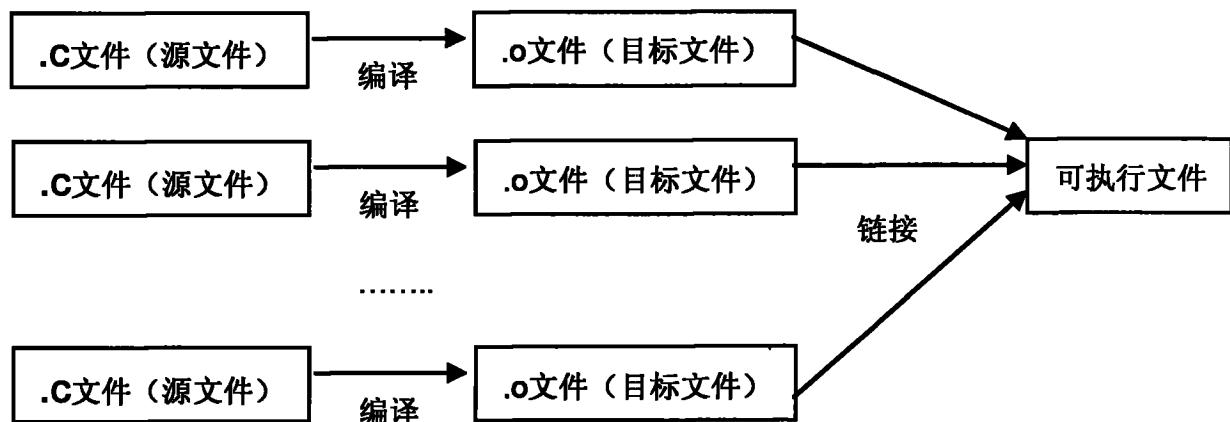


图 9

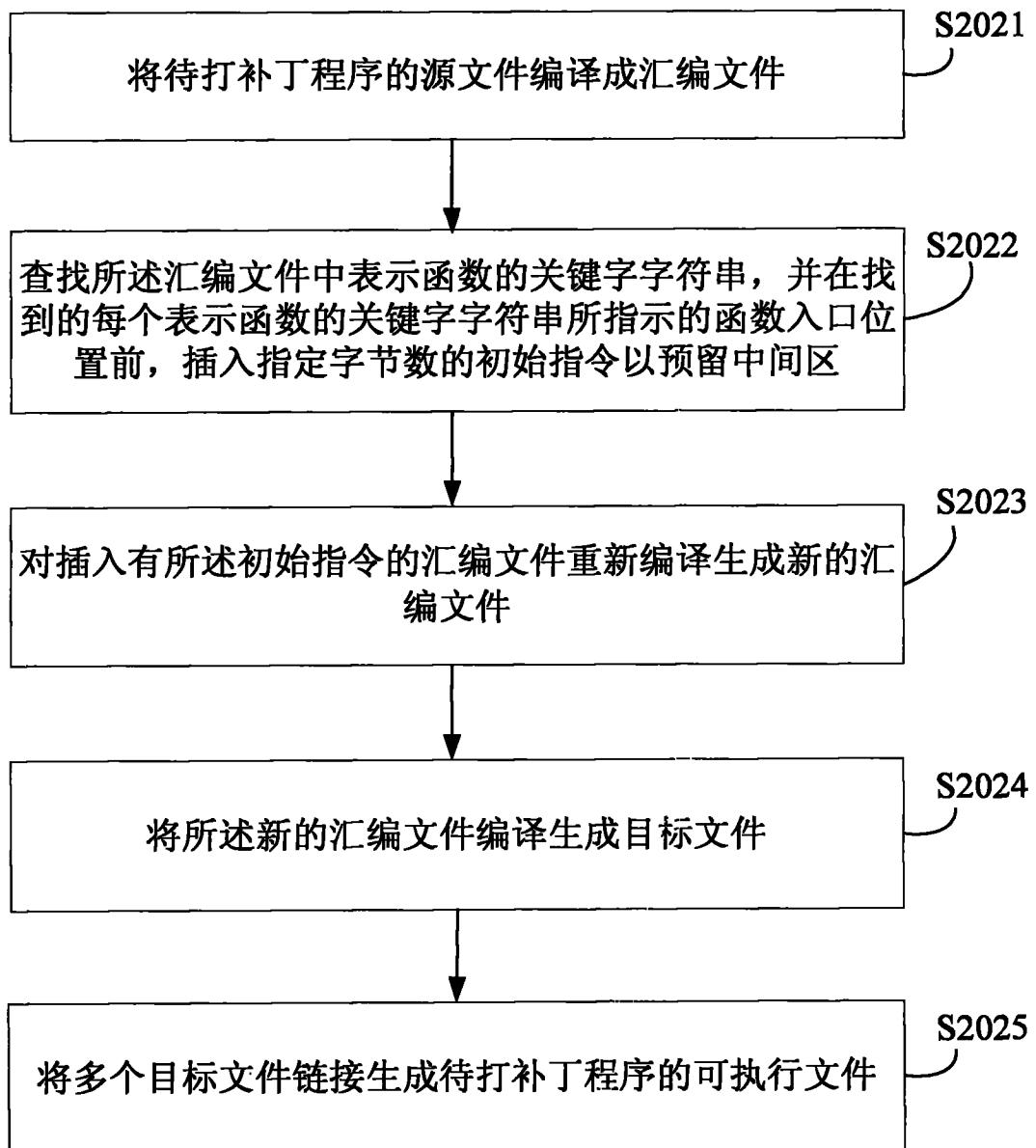


图 10

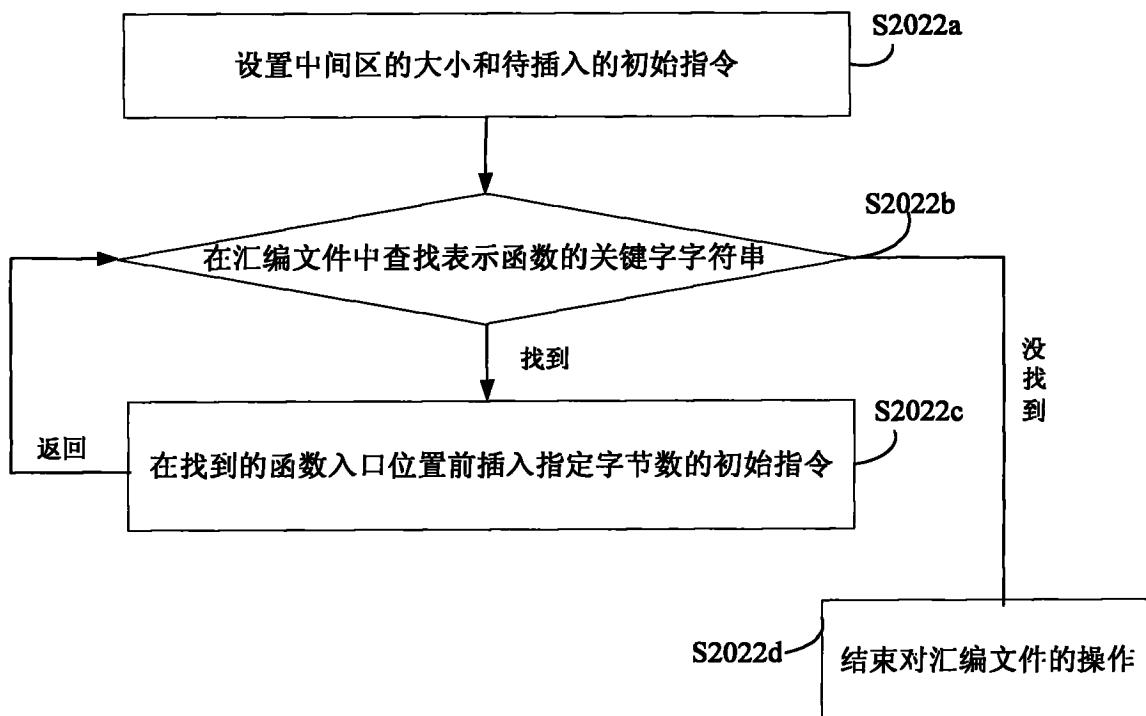


图 11

```
----- .text
.globl test
.type test, @function
test:
    pushl %ebp
    movl %esp, %ebp
    subl $8, %esp
    movl $.LC0, (%esp)
    call vos_printf
    movl $.LC1, (%esp)
    call vos_printf
    movl $.LC2, (%esp)
    call vos_printf
    movl $.LC3, (%esp)
    call vos_printf
    movl $.LC4, (%esp)
    call vos_printf
    leave
    ret
.size test, .-test
.section .rodata
.LC5:
.string "\r\nthe value of b is [%#x]"
.text
.globl test1
.type test1, @function
```

图 12a

```
.text
byte 0x90
byte 0x90
byte 0x90
byte 0x90
byte 0x90
byte 0x90

.globl test
.type test, @function
test:
    pushl %ebp
    movl %esp, %ebp
    subl $8, %esp
    movl $.LC0, (%esp)
    call vos_printf
    movl $.LC1, (%esp)
    call vos_printf
    movl $.LC2, (%esp)
    call vos_printf
    movl $.LC3, (%esp)
    call vos_printf
    movl $.LC4, (%esp)
    call vos_printf
    leave
    ret
.size test, .-test
.section .rodata
.LC5:
.string "\r\nthe value of b is [%#x]"
.text
byte 0x90
byte 0x90
byte 0x90
byte 0x90
byte 0x90
byte 0x90

.globl test1
.type test1, @function
test1:
    pushl %ebp
    movl %esp, %ebp
```

图 12b

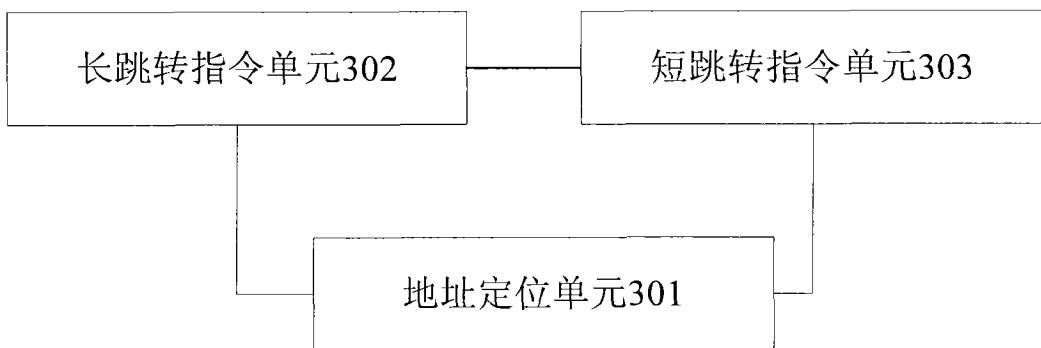


图 13