doi:10.3772/j.issn.1002-0470.2015.06.001

二进制翻译系统中信号处理机制的研究①

远 翔②*** 武成岗③* 王振江*

(*中国科学院计算技术研究所计算机体系结构国家重点实验室 北京 100190)

(**中国科学院大学 北京 100190)

摘 要 考虑到用户级二进制翻译(BT)系统需要支持在应用程序特别是多线程程序中广泛使用的信号机制,研究了用户级BT系统中的信号处理问题。详细分析了BT系统在处理信号时存在的各种问题,在此基础上设计了一种新的BT系统执行框架,用以降低用户级BT系统由于同时处理自己的信号和访客程序的信号造成的信号处理复杂度;提出了BT系统的信号隔离机制,用以避免BT系统处理信号时出现信号丢失和死锁问题,有效提高信号处理的健壮性。该机制的性能和有效性已通过测试试验得到验证。

关键词 用户级二进制翻译(BT),信号处理,多线程框架,信号隔离

0 引 言

二进制翻译(binary translation,BT)技术通过代码翻译使一种指令集体系结构(instruction set architecture,ISA)的可执行程序可以移植到另一种指令集体系结构上执行,以促进新指令集体系结构处理器的推广和发展。可执行程序原先所在的平台通常被称作访客平台,它所移植到的平台被称作宿主平台。这类典型的系统有FX!32[1,2]、UQBT[3,4]、IA32-EL[5]、QEMU[6]、Aries[7]等。当访客平台和宿主平台的指令集体系结构(ISA)相同时,该项技术常被用于通过插桩收集程序的剖析(profile)信息、进行程序分析和错误检测,这方面典型的系统有Pin[8]、Valgrind[9]、DynamoRIO[10]等;该项技术还被用于动态优化和提升程序的性能,这方面典型的系统有Dynamo[11]、ADORE[12]等。

二进制翻译(BT)系统可以分为系统级和用户级两种[13]。在实际应用中,用户级BT系统移植的程序往往利用信号进行线程(或进程)间通信。信号是软件层次上对于中断的模拟[14]。它被Unix以及其它POSIX兼容的操作系统所使用,如Linux、Mac OS X、Solaris、z/OS等。信号机制可以是异步的,一个线程不需要等待信号的到达。当它收到信号时,操作系统会自动执行它所注册的信号处理程序。信号的这种特性使得它具有很强的灵活性,从而被实际程序尤其是并行程序广泛使用。本文调查了Linux系统下常用的46个应用程序(如Apache Httpd、MySQL、Transmission、PBZIP2等),其中37个使用了信号机制。信号的这种异步特征给程序执行带来不确定性,这使得由于信号导致的错误难以出现以及调试。在广泛使用信号的并行程序中,由于并行程序本身存在不确定性,因而使得该问题更加严重。因此对于用户级BT系统(特别是商业化的BT系统)而言,要实现并行程序的移植就必须对信号提供有效的支持。

用户级BT系统(本文简写为BT系统)在宿主操作系统之上仿真执行访客应用。和通常的应用程

① 国家自然科学基金(61303052,61332009,61303051,60925009),863计划(2012AA010901),国家自然科学基金创新群体(61221062)和973计划(2011CB302504)资助项目。

② 男,1984年生,博士生;研究方向:动态编译和并行程序错误检测;E-mail:yuanxiang@ict.ac.cn

③ 通讯作者,E-mail:wucg@ict.ac.cn

(收稿日期:2014-12-29)

序相比,BT 系统中进行信号处理更加复杂的原因主要有两点:(1) BT 系统在支持访客程序执行时会同时收到访客程序和 BT 系统自身的信号,这使得 BT 系统需要同时处理这两种的信号。(2)对于使用信号的程序而言,程序员可以通过合理的设计来保证正确使用信号,但是 BT 系统在移植访客程序时,无法预测在何时收到何种访客程序的信号,而这些信号会对 BT 系统的执行产生干扰。反之,BT 系统的信号也会干扰访客程序生成的本地码的执行。这种相互干扰大大增加了信号处理的复杂度。但是,目前对信号的处理主要针对某几个特定的问题,没有对 BT 系统中的信号处理进行全面的分析,同时目前 BT 系统的信号处理可能带来信号丢失、死锁等问题,它们不能保证信号处理在 BT 系统,特别是商业化 BT 系统中的正确性。因此,本研究分析了 BT 系统中信号处理的难点,提出了一种新的 BT 系统执行框架,以简化 BT 系统中信号处理,同时提出了 BT 系统的信号处理机制,以避免信号处理时出现信号丢失和死锁等问题。

1 信号机制给 BT 系统带来的问题

1.1 信号机制简介

信号根据来源的不同可以分为两类[7,11]:一是某个线程运行过程中发生异常(如除零、非法内存访问等)而引发的同步信号;二是其它线程通过某些系统调用而发来的异步信号。当某个线程收到信号时,操作系统则会暂停这个线程,转而执行对应的信号处理程序。程序在执行过程中,可以根据需要来设置信号屏蔽字来暂时屏蔽某些信号。被屏蔽的信号被操作系统保存,不会丢失。在解除屏蔽之后,程序就可以收到这些信号。

1.2 问题分析

访客程序由 BT 系统支撑在宿主平台运行。如果它使用了信号,一般情况下,它会通过系统调用向操作系统注册相应信号处理程序。但是它注册的是访客程序中信号处理程序(称为访客信号处理程序)的入口地址。那么当访客程序收到信号时,宿主操作系统会直接执行访客信号处理程序。这使得该信号处理程序就脱离了二进制翻译的支撑。因此,BT 系统在访客程序注册信号处理程序时,应该注册一个自己的信号处理程序,由其仿真执行访客信号处理程序。

在 BT 系统中,二进制翻译本身会给信号处理带来新的问题,而且 BT 系统本身也会使用信号,这些信号和访客程序的信号会相互干扰。因此,本文将从这两个角度对 BT 系统中的信号处理进行详细地分析。

1.2.1 二进制翻译的角度

(1)访客、宿主平台信号的一致性

访客程序在执行时,它的各个线程可以使用信号进行通信。同时,它还可以利用信号与其它宿主程序以及宿主操作系统进行通信。但是,不同平台的信号并不是完全相同的。每一个信号都有名称和序号。在不同的平台,相同名称的信号序号可能不同,相同名称信号的功能可能不同,不同平台的信号种类和数量可能不同。因此,BT 系统需要维护平台间信号的一致性。

(2)构建访客信号处理程序的栈帧

在执行信号处理程序之前,操作系统首先会构建它的栈帧,其中包括线程收到信号时的上下文(如寄存器值等)等信息。但是,访客程序实际上在宿主平台执行,它收到信号时,操作系统构建的是宿主信号处理程序的栈帧。为了保证访客信号处理程序的正确执行,它的栈帧需要 BT 系统构建。BT 系统创建该栈帧时应该保证其中线程的上下文和访客程序在访客平台执行某条指令结束时的上下文一致。但是 BT 系统的指令翻译可能将一条访客指令翻译为若干宿主指令,同时它的优化可能会调整指令顺序。因此,BT 系统在构建访客栈帧时需要保证其中线程上下文的一致性。

1.2.2 避免干扰的角度

BT 系统本身也会使用信号进行通信。这些信号和访客程序的信号同时存在,会干扰 BT 系统或访客程序的执行。BT 系统自身和访客程序均可以使用信号进行通信,如果它们使用相同的信号会难以区分。另外它们也会使用信号屏蔽字来屏蔽不同的信号。因此,BT 系统需要保证它使用的信号以及

信号屏蔽字和访客程序互不干扰。

部分系统调用(如 futex( )、对管道的 read( )等)会使得调用它的线程进入阻塞状态。当一个被系统调用阻塞的线程收到信号时,信号会使得该线程提前从阻塞状态中退出。在通常的应用程序中,程序员可以通过良好的设计避免信号干扰此类系统调用。但是在 BT 系统中,访客程序不知道 BT 系统的存在;BT 系统也不知道访客程序可能使用的信号。这样 BT 系统需要保护系统调用免受信号的干扰。

另外,还要考虑代码可重入问题。对于 BT 系统而言,支撑访客程序执行的代码是系统的主体部分,而且这部分代码伴随着访客程序的执行而不断被调用。BT 系统在执行这些代码时,能够被访客程序的信号中断,这就要求 BT 系统中支撑访客程序执行的代码是可重入的。按照文献[15]的要求,可重入代码只能处理由调用者提供的数据,不能调用单实例模式的锁,不能调用其它不可重入代码。但是 BT 系统需要调用不可重入的库函数(如 printf( )、malloc( )、free( )等[16]),同时为了支持多线程的访客程序,不可避免地需要使用全局变量和锁。这使得 BT 系统很难保证上述代码可重入。

1.2.3　现有 BT 系统信号处理的问题

目前已有的 BT 系统[5-9,11,17]在进行信号处理上注重如下三点:(1)在注册信号处理程序时注册 BT 系统自己的信号处理程序;(2)在收到同步信号时,立刻翻译执行访客信号处理程序;(3)在收到异步信号时,将信号暂时缓存,等到执行 BT 系统本身代码时再进行处理。这使得上述 BT 系统在仿真执行访客程序时可能收到 BT 系统自己的信号,在执行 BT 系统代码时可能收到访客程序的信号。因此 BT 系统需要经过仔细设计来避免 1.2.1 和 1.2.2 节中的问题出现导致错误、增加设计难度。

这些 BT 系统没有讨论信号对系统调用的干扰问题,而且 BT 系统[6,7,9,11,17]采用如下方案处理可重入问题:BT 系统在收到信号之后,将信号暂时缓存,当执行到某些特定的点之后,再翻译执行访客信号处理程序。这样,BT 系统只要保证在这些点上可重入即可。采用这种方案,如果访客程序在收到信号后立即退出或者执行阻塞的系统调用,上述信号处理机制会因为 BT 系统不能及时执行访客信号处理程序,而导致访客程序发生错误。图 1(a)中线程 T0 在检查缓存信号之后再次收到信号,而它在下一次检查缓存信号之前退出,这使得 T0 虽然收到了信号但是实际上并没有执行对应的处理程序。这会使得程序执行发生错误。图 1(b)线程 T1 收到信号并缓存,然后立即执行系统调用 A,被阻塞。T0 也由于执行系统调用 B 被阻塞。但是,唤醒 T0 的操作位于信号处理程序中,由于信号被缓存没有执行。这样使得 T0 和 T1 发生死锁。

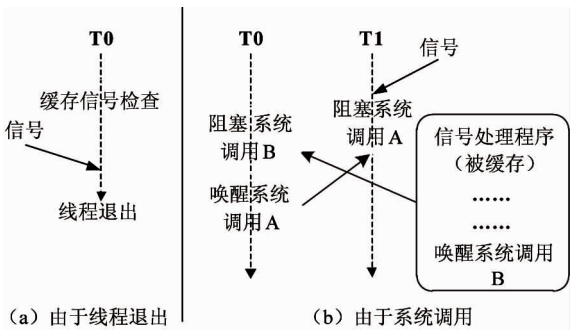

图 1　缓存信号导致错误

2　BT 系统中的信号处理

第 1 节的分析说明 BT 系统中信号处理复杂的原因是:BT 系统在执行自身代码时能够收到访客程序的信号,而仿真执行访客程序时能够收到 BT 系统的信号。这些信号是未知的,它们会干扰 BT 系统和访客程序的执行。如果 BT 系统在执行自身代码时只收到 BT 系统的信号,在执行本地码时只收到访客程序的信号,那么就能够避免这种干扰,降低 BT 系统中信号处理的难度。

针对这个问题,本文提出了一种新的 BT 系统执行框架,并设计了信号隔离机制来处理信号。

2.1　BT 系统执行框架

图 2 是该执行框架的示意图,其中 BT 系统采用多线程执行的形式,它的线程分为 BT 线程和模拟线程两类。BT 线程负责执行 BT 系统自身的代码。而模拟线程负责执行访客程序翻译得到的本地码,和访客程序的线程一一对应。当访客程序创建

新线程时,BT 系统也新创建一个模拟线程。BT 系统启动时,它在 BT 线程中装载访客程序,并从访客程序的入口地址开始进行代码翻译。当需要执行本地码时,BT 线程通知模拟线程,由模拟线程实际执行本地码。

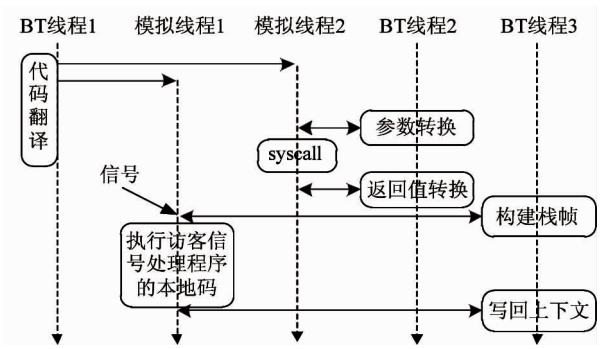

图 2 BT 系统执行框架

而模拟线程在执行本地码时,会调用 BT 系统的代码,这可以分成三种情况[1-7]:

(1)系统调用:访客程序在 BT 系统支持下运行时,会使用访客系统调用的参数调用宿主系统调用。这样在执行系统调用前后,需要 BT 系统将参数和返回值在访客平台以及宿主平台格式之间转换。(图 2 中模拟线程 2 和 BT 线程 2 所示)。

(2)执行信号处理程序:在执行访客信号处理程序前,需要 BT 系统根据模拟的访客平台上下文(包括寄存器值、堆栈等)构建它的栈帧。当访客信号处理程序结束之后,BT 系统需要将它对栈帧的修改写回(图 2 中模拟线程 1 和 BT 线程 3 所示)。

(3)翻译访客代码:当模拟线程执行时发现未翻译的访客代码时,它需要调用 BT 系统进行翻译(图 2 中 BT 线程 1、模拟线程 1 和模拟线程 2 所示)。

在上述三种情况中,模拟线程为了通过 BT 线程调用 BT 系统的代码,情况(1)、(2)可以采用图 3(a)的流程,即:模拟线程首先根据 BT 代码的要求准备参数,再通知 BT 线程并暂停执行;BT 线程完成模拟线程所需的操作之后,通知模拟线程;模拟线程得到 BT 线程的执行结果之后,再继续执行。但是对于情况(3),由于本地码下一次执行时不再需要对访客代码进行翻译,因此 BT 系统需要采用图 3(b)的流程。此时 BT 线程完成对访客代码的翻译之后,还应该修改模拟线程当前正在执行的本地码,使它下次执行时能够直接跳转并执行下一段本地码。除此之外,对于其它的情况也可以采用类似的方法处理。

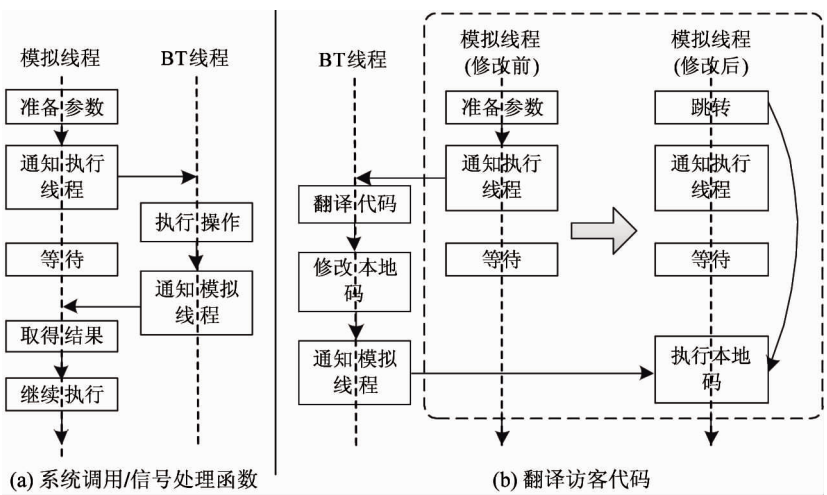

图 3 执行切换

在上述 BT 系统执行框架中,BT 系统的代码均由 BT 线程执行,本地码均由模拟线程执行。在此基础上,本文设计了 BT 系统的信号处理机制。

2.2 信号处理机制

本节首先介绍信号隔离,然后按照信号的工作过程,逐步进行介绍。

2.2.1　信号隔离

在第2.1节的执行框架基础上,本文设计了信号隔离机制,保证模拟线程仅收到访客程序的信号,而BT线程仅收到BT系统自身的信号。

信号可以被发送给线程或进程。发送给线程的信号只能被该线程接收,而发送给进程的信号可以被该进程内任意一个未屏蔽该信号的线程接收。对于发送给线程的信号,BT系统自身可以保证仅在BT线程之间使用信号,而访客程序在宿主平台执行时不知道BT系统的存在,因此它不会向BT线程发送信号。对于发送给进程的信号,它可能被BT线程或模拟线程接收,本文通过信号屏蔽来进行处理,即:BT线程屏蔽访客程序使用的信号,模拟线程屏蔽BT系统使用的信号。但是BT系统不知道访客程序使用哪些信号。因此,BT系统需要动态检查访客程序执行的信号相关系统调用,并动态更新BT线程的信号屏蔽字。

2.2.2　注册信号处理程序

BT系统注册信号处理程序包括两步:(1)系统初始化时,为BT系统自身使用的信号注册对应的信号处理程序;(2)拦截访客程序中注册信号的系统调用。当访客程序注册信号处理程序时,记录它的信号处理程序的地址,并将其替换为BT信号处理程序的地址。由于信号隔离机制,只有模拟线程能够收到访客信号并执行BT信号处理程序。它的功能是:利用BT线程构建访客信号处理程序的栈帧并对其进行翻译;执行访客信号处理程序生成的本地码;利用BT线程进行栈帧写回。

2.2.3　发送和接收信号

(1)维护信号一致性

由于相同名称的信号在不同平台序号可能不同,而程序在发送和接受信号时使用的是信号的序号。因此BT系统需要在访客程序发送信号之前将访客信号转换为宿主信号,同时在其收到信号之后再进行反向转换。这种转换可以通过一个映射表来实现。如果访客程序使用的某个访客平台信号在宿主平台不存在,为了使访客程序正常工作,这需要BT系统进行模拟。如:MIPS平台存在非对齐内存访问的信号,为了在x86平台模拟该信号,BT系统需要检查访客程序的每一次访存的地址,这会影响BT系统的性能。不过,经过调查我们看出,绝大部分实际应用程序不存在这种情况。

(2)避免干扰系统调用

信号隔离使得BT系统自身在执行系统调用时,不需要考虑访客信号带来的干扰。这样BT系统的设计者就可以根据BT系统自身使用的信号,对可能被干扰的系统调用进行保护或采取相应的补偿措施(如重新启动被干扰的系统调用)。同样BT系统的信号也不会干扰访客程序系统调用的执行。这样,本文的信号隔离机制有效地降低了BT系统信号处理时保护系统调用的难度。

2.2.4　执行访客信号处理程序

(1)可重入问题

在BT系统中,信号带来的可重入问题复杂的主要原因是:BT系统中支撑访客程序执行的代码需要满足可重入的要求。而本文的信号隔离机制使得BT系统在执行上述代码时,不会收到访客程序的信号。这样,BT系统自身代码只需要保证对于BT系统自身的信号可重入即可。本文的机制使得当前BT系统中处理可重入而引入的问题(1.2.3节所述)不再存在,提高了BT系统信号处理的健壮性。

(2)获得访客程序上下文

当模拟线程收到访客程序的信号时,按照收到信号类型的不同,BT系统需要采用不同的方法处理这个问题。

(a)同步信号。本文采用了文献[5]的方法:在生成某条访客指令的本地码时,尽可能将修改访客程序上下文的指令放到可能产生同步信号的指令(如:访存指令)之后。对于无法达到该要求的指令,BT系统在翻译指令前保存此时的访客上下文。这种方法依然不是完备的。如果确实需要保证BT系统在收到同步信号时访客程序上下文的一致性,可以在每一条访客程序指令之后保存上下文并禁用会改变访客程序上下文的优化,如指令调度等。但是,这样会严重影响BT系统的执行速度。

(b)异步信号。BT系统在翻译访客程序时,记录“一致点”的位置。当BT系统支撑访客程序执行到“一致点”时,访客程序的上下文和它在访客平台

执行时某条指令结束时上下文一致。“一致点”的位置往往是访客程序中的一条指令、一个基本块或者若干个基本块结束时对应的本地码位置。若模拟线程收到信号的位置不是“一致点”,那么 BT 信号处理程序继续执行本地码,直到下一个“一致点”时再执行访客信号处理程序。

图 4 是处理的具体过程。访客程序的 4 条指令经过翻译变成 9 条宿主平台指令,其中下划线标记的指令为“一致点”。模拟线程在执行完 tinst _ 1 时收到信号,此时并不位于“一致点”。因此,BT 信号处理程序将 tinst _ 2 和 tinst _ 3 复制到其内部缓存中,并执行这两条指令。之后,BT 信号处理程序根据执行的结果构建访客程序上下文。当信号处理结束之后,访客程序从 tinst _ 4 处恢复执行。

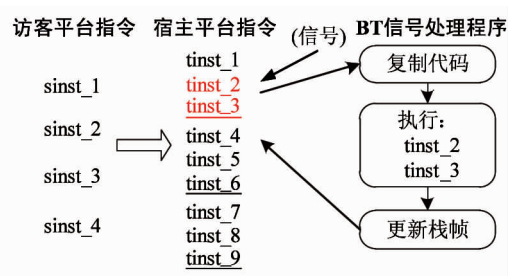

图 4 得到完整的访客程序上下文

3 相关工作

Intel 开发的二进制插桩工具 Pin[8,17] 对于信号采用了如下的处理方法:当 Pin 收到信号之后,它将这个信号缓存到一个队列中,并断开本地码之间的直接跳转。这样,当 Pin 在执行本地码时收到信号,它能够尽快返回 Pin 系统代码。之后,Pin 为访客信号处理程序构建栈帧,并翻译执行。

HP 的 Aries[7] 系统实现了从 PA-RISC 到 IA-64 的二进制翻译。Aries 将收到的异步信号缓存,当它能够提供完整的访客程序上下文时再进行处理。Aries 在收到同步信号之后,会立即执行访客信号处理程序。另外,动态二进制插桩框架 Valgrind[9] 和多源多平台的 BT 系统 QEMU[6] 也是采用了同样的方法处理访客程序收到的信号。

动态优化系统 Dynamo[11] 的信号处理主要针对如何获得完整的访客程序上下文进行。对于异步信号,Dynamo 采用和 Aries 同样的方法进行处理。对于同步信号,Dynamo 采用去优化的方法。它将优化分为两种:保守优化和激进优化,其中前者不会破坏访客程序的上下文。Dynamo 通常对访客程序生成激进优化后的本地码。当它发现可能产生同步信号的指令序列时,对这样的指令序列生成保守优化的本地码。这样,使得访客程序产生同步信号时,Dynamo 能够得到完整的访客程序上下文。

上述二进制翻译系统的相关文献对信号处理主要关注于如何获得完整的访客程序上下文,而没有说明如何解决 BT 系统处理信号时存在的其它问题,如维护信号一致性、避免打断系统调用等。上述系统对异步信号的缓存处理可以避免可重入问题,但是会带来死锁或者信号丢失。而本文从信号的机制出发,分析了线程在使用信号进行通信的各个步骤在 BT 系统中可能存在的问题。为了更好地在 BT 系统中进行信号处理,本文提出了 BT 系统新的执行框架,实现了信号隔离并提出了更完整的 BT 系统信号处理机制。

4 试 验

BT 系统 DigitalBridge 实现了本文提出的信号处理机制。DigitalBridge 将 X86/Linux 下的可执行程序翻译到 MIPS/Linux 平台下执行。

为了进行评估,本文将 DBT 系统和 Pin 进行比较。Pin[8] 是 Intel 公司组织为了进行程序行为分析而开发的二进制插桩工具,该系统自发布以来得以广泛应用。Pin 采用二进制翻译的方式实现同平台的程序插桩,因而也会面临信号支持方面的问题。这样,本文可以通过和 Pin 进行对比来评估本文提出的方法。

本文利用 Open POSIX Test Suite[18] 和常见的应用程序 Apache HTTPD[19] 和 Transmission[20] 来评估本文提出的方法。测试环境如表 1 所示。

表 1　测试平台配置

	MIPS	x86
CPU/ISA	MIPS III,单核	Xeon E7-4807,6 核
主频	1GHz	1.87GHz
Cache	L1: 64KB * 2	L1: 2 * 24KB * 6
	L2: 512KB	L2: 256KB * 6
		L3: 18MB
内存	1G	8G
操作系统	Linux 2.6.18	Linux 2.6.39
编译器	GCC 4.3.1	GCC 4.6.1

4.1　功能性测试

对于 BT 系统而言,它应该保证访客程序在宿主平台运行时的行为和访客程序在访客平台运行时的行为是一致的。

Pin 采用二进制翻译的方法对访客程序进行插桩。为了能够精确地得到访客程序的信息,它也应该保证插桩代码不会改变访客程序本身的行为。

本文通过开放测试集以及实际应用程序来比较本文的信号处理机制以及 Pin 中的信号处理机制能否保证访客程序行为的一致性。

4.1.1　测试 Open POSIX Test Suite

Open POSIX Test Suite 是一个针对 POSIX API 的测试包。它针对线程、时钟和定时器、信号、消息队列以及信号量进行测试。目前它的最新版本 1.5.2中包括 1711 个测试用例,其中需要处理信号的测试用例 700 个。试验将这 700 测试用例各执行 100 遍,对比执行结果。通过试验证明,采用本文提出的信号处理方案的 DigitalBridge 系统没有因为信号发生错误。

而 Pin 在测试过程中,有 33 个例子产生了死锁,如表 2 所示。为了避免干扰,测试过程中没有使用任何 PinTool。“直接执行”表示该错误使用 Pin 直接进行测试就会出现。“插入延迟”表示该错误很难出现,本文在测试时在测试程序中插入延迟来提高其出现几率。表 2 的测试用例的特征是通过信号进行线程间同步操作。通过对这些测试用例进行分析,发现这种死锁是由于 Pin 对信号进行缓存处理(如图 1 所示)引起的。当信号被缓存之后,由于收到信号的线程退出或者被阻塞,使得访客信号处理程序没有被执行,从而导致死锁产生。Pin 对信号进行缓存是为了处理可重入问题,而本文的方案通过信号隔离避免了可重入问题的影响,不需要对收到的信号进行缓存,从而避免了这种错误的发生。

表 2　Pin 的 Open POSIX Test Suite 测试结果

子测试包	错误数	详情	备注
pthread _ create	1	14-1. test	直接执行
pthread _ detach	1	4-3. test	
pthread _ join	1	6-3. test	
pthread _ mutex _ lock	1	5-1. test	
sigaction	26	16- * . test	
pthread _ barrier _ wait	1	3-2. test	插入延迟
pthread _ rwlock _ rdlock	1	4-1. test	
pthread _ rwlock _ wrlock	1	2-1. test	

4.1.2　测试 Apache HTTPD 和 Transmission

Apache HTTP Server 是一个开放源码的 Web 服务器,由于其多平台和安全性被广泛使用,是目前最流行的 Web 服务器端软件之一。Transmission 是一个开源的比特流(BitTorrent)下载客户端,它支持 Linux、Mac OS X、Solaris 等操作系统。Ubuntu Linux 选择了它作为默认的比特流下载器。

Apache HTTPD 和 Transmission 使用信号来终止程序的运行。这两个程序包含了若干个线程,其中 Apache HTTPD 的创建者线程和 Transmission 的 DHT 线程会在程序执行的过程中退出。在 Pin 的处理方案中,如果 Apache HTTPD 的创建者线程或 Transmission 的 DHT 线程在退出前夕收到终止信号,那么会导致这个信号丢失(如图 5 所示)。

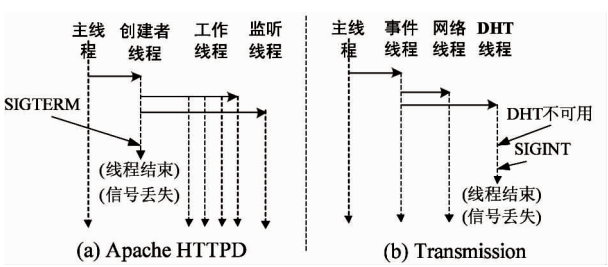

图 5　Pin 导致信号丢失

本文测试的方法是通过插桩,使得 Apache HTTPD 的创建者线程和 Transmission 的 DHT 线程在执

行 exit( ) 前休眠一段时间。在这段时间内,通过 tkill( ) 向上述线程发送终止信号来进行测试。

在对 Pin 进行测试过程中,如果上述线程在休眠期间收到终止信号,那么当休眠结束时,应用程序会继续执行。该过程中,线程在收到信号之后立即退出,这使得 Pin 的信号处理机制还没来得及执行对应的信号处理程序。这使得这些应用程序无法正常退出。本文的信号处理方案使得 BT 系统在收到信号之后可以立即执行对应的信号处理程序,避免了上述问题,保证了应用程序执行的正确性。

4.2 性能测试

为了比较本文的信号处理机制的性能,本文在 DigitalBridge 中实现了 Pin 的信号处理方案。

本文从 Open POSIX Test Suite 中选出 18 个测试用例进行性能测试。这些测试用例的特征是在执行过程中不断发送和接收信号,它们在执行 1 秒之后退出。为了测试性能,本文将它们修改为处理 1000 个信号之后退出。为了使得 Pin 和本文的信号处理机制能够进行比较,本文按照文献[8]和[17]的说明,在 DigitalBridge 系统中实现了 Pin 的信号处理机制。对上述 18 个测试用例,每个执行 10 次得到的平均性能数据如图 6 所示。本文机制的性能和 Pin 差别不大。本文方案的额外开销主要由模拟线程和 BT 线程的交互引入。上述交互仅发生在代码翻译、执行系统调用以及执行信号处理程序时。BT 系统中对于访客平台代码的翻译一般仅执行一次,对性能影响不大。而执行系统调用和信号处理程序本身开销比较大,本文在这两种情况下引入的开销不明显。实际应用在执行过程中并不会像这些测试程序一样频繁地发送和接收信号。因此,对于实际应用而言,本文信号处理机制对于性能的影响更小。

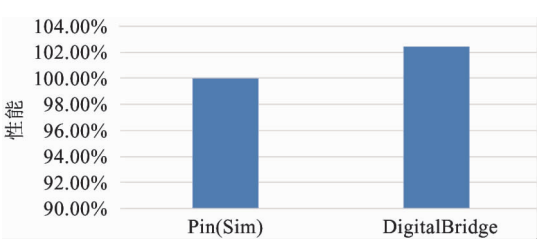

图 6 信号处理性能

5 结 论

对于用户级二进制翻译系统而言,能够正确地支持信号机制十分重要。因此,本文通过对信号机制的各个步骤进行分析,从二进制翻译条件下支持信号机制需要解决的问题以及信号对 BT 系统的影响两个方面查找 BT 系统中支持信号所需要解决的问题。为了更好地在 BT 系统中进行信号处理,本文提出了 BT 系统新的执行框架和信号隔离机制,以降低 BT 系统信号处理机制解决可重入以及信号干扰系统调用等问题的难度。通过常用测试集以及实际应用程序对其进行了测试,证明与以 Pin 为代表的其它 BT 系统的信号处理机制相比,本文方法有效地避免了信号丢失以及死锁等问题,提高了 BT 系统的健壮性,同时基本没有影响 BT 系统的性能。

同步信号是由程序执行的异常产生的,BT 系统需要在收到同步信号时立即执行访客信号处理程序。目前 BT 系统在处理这种情况时,为了保证访客上下文的一致性,会产生较大的性能下降。因此,如何降低该情况时对性能的影响,是将来需要进一步研究的内容。

参考文献

[1] Chernoff A, Herdeg M, Hookway R, et al. FX!32: a profile-directed binary translator. *IEEE Micro*, 1998, 18(2): 56-64

[2] Hookway R, Herdeg M. Digital FX!32: Combining emulation and binary translation. *Digital Technical Journal*, 1997, 9(1): 3-12

[3] Cifuentes C, Van Emmerik M. UQBT: Adaptable binary translation at low cost. *Journal Computer*, 2000, 33(3): 60-66

[4] Cifuentes C, Van Emmerik M, Ung D, et al. Preliminary experiences with the use of the UQBT binary translation framework. In: Proceeding of the Workshop on Binary Translation. NewPort Beach, Technical Committee on Computer Architecture Newsletter. 1999. 12-22

[5] Baraz L, Devor T, Etzion O, et al. IA-32 Execution layer: a two-phase dynamic translator designed to support IA-32 applications on Itanium-based systems. In: Pro-

ceedingof 36th International Symposium on Microarchitecture, Washington, DC, USA, 2003. 191-204

[6] Bellard F. QEMU, a fast and portable dynamic translator. In: Proceedings of the USENIX Annual Technical Conference, Berkeley, USA, 2005. 41-46

[7] Zheng C, Thompson C. PA-RISC to IA-64: Transparent execution, no recompilation. *Journal Computer*, 2000, 33(3): 47-52

[8] Luk C, Cohn R, Muth R, et al. Pin: building customized program analysis tools with dynamic instrumentation. In: Proceedings of the Conference on Programming Language Design and Implementation, New York, USA, 2005. 190-200

[9] Nethercote N, Seward J. Valgrind: a framework for heavyweight dynamic binary instrumentation. In: Proceedings of the Conference on Programming Language Design and Implementation, San Diego, USA, 2007. 89-100

[10] Bruening D. Efficient, Transparent, and Comprehensive Runtime Code Manipulation. [Ph. D dissertation], MIT, 2004

[11] Bala V, Duesterwald E, Banerjia S. Dynamo: a transparent dynamic optimization system. In: Proceeding of the Conference on Programming Language Design and Implementation, New York, USA, 2000. 1-12

[12] Lu J W, Chen H, Fu R, et al. The performance of runtime data cache prefetching in a dynamic optimization system. In: Proceeding of 36th International Symposium on Microarchitecture, Washington, USA, 2003. 180-189

[13] Smith J, Nair R. Virtual Machine: Versatile Platforms for Systems and Processes. Elsevier, 2005

[14] 毛德操,胡希明. Linux 内核源代码情景分析. 浙江:浙江大学出版社,2001

[15] IBM. Writing Reentrant and Thread-Safe Code, from AIX 5L Version 5.3General Programming Concepts: Writing and Debugging Programs, 8th Edition. 2010 [2012-12-31]. http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.genprogc/doc/genprogc/genprogc.htm

[16] IEEE. IEEE Std 1003.1, Version 4. 2008. http://www.unix.org/version4/

[17] Hazelwood K, Lueck G, Cohn R. Scalable support for multithreaded applications on dynamic binary instrumentation systems. In: Proceeding of the International Symposium on Memory Management, New York, USA, 2009. 20-29

[18] Open POSIX Test Suite. http://posixtest.sourceforge.next/

[19] Apache HTTPD. http://www.apache.org/

[20] Transmission. http://www.transmissionbt.com/

Research on signal handlingin binary translators

Yuan Xiang[*,**], Wu Chenggang[*], Wang Zhenjiang[*]

([*]State Key Laboratory of Computer Architecture, Institute of Computing Technology, CAS, Beijing 100190)

([**]University of Chinese Academy of Sciences, Beijing 100190)

Abstract

The signal handling in user-level binary translation (BT) systems was studied with the consideration that user-level BT systems must support the signal mechanism widely used in applied programs, especially in the multi-threaded ones. Based on the detailed analysis of the various problems in BT systems' signal handling, a new execution framework for BT systems was designed to lower the signal handling complexity caused by simultaneous handling of their own signals and the signals generated by guest programs, and a signal isolation mechanism was proposed to avoid the problems of signal losing and deadlock in signal handling to effectively improve the robustness of signal handling. The performance and the effectiveness of the proposed mechanism were verified by test.

Key words: user-level binary translation(BT), signal handling, multi-thread framework, signal isolation