ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.5, May 2006, pp.1232−1240
DOI: 10.1360/jos171232

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于 WCET 分析的实时系统轨迹获取技术*

王 馨, 姬孟洛, 王 戟+, 齐治昌

(国防科学技术大学 计算机学院,湖南 长沙 410073)

Trace Acquirement Technology of Real-Time Systems Based on WCET Analysis

WANG Xin, JI Meng-Luo, WANG Ji+, QI Zhi-Chang

(School of Computer, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: Phn: +86-731-4573637, Fax: +86-731-4575802, E-mail: jiwang@nudt.edu.cn, http://www.nudt.edu.cn

Wang X, Ji ML, Wang J, Qi ZC. Trace acquirement technology of real-time systems based on WCET analysis. *Journal of Software*, 2006,17(5):1232−1240. http://www.jos.org.cn/1000-9825/17/1232.htm

Abstract: Timing behaviors are crucial for real-time systems. In order to weaken or even remove the timing impact of the inserted assertions during testing, a new monitoring schema is proposed, which has little time intrusiveness than the software monitoring and supports to test the system completely. Furthermore, a WCET (worst-case execution time) analysis based on time compensation method is presented, which corrects the recorded time of events based on the accurate execution time analysis of assertions.
Key words: real-time system; test oracle; WCET (worst-case execution time) analysis; program monitoring

摘 要: 时序约束是判断实时系统运行是否正确的重要规约.为了减小测试时由于对系统进行插装而产生的对实时系统行为的影响,提出了一种混合式监控方法.它对系统的时间干扰比纯软件方式小,并支持对系统的完全测试.此外,还提出一种基于 WCET(worst-case execution time)分析技术的目标系统时间补偿方法,在精确地计算插入断言对目标系统的时间影响基础上,给出时间补偿.
关键词: 实时系统;测试预言;WCET(worst-case execution time)分析;程序监控
中图法分类号: TP316 文献标识码: A

实时系统目前越来越多地应用于人们生活的各个方面,特别是在航空航天、医疗监控、军事指挥和武器装备控制中.在这些领域内,对实时系统的安全性和实时性要求是非常严格的,一旦软件控制出现问题,其后果可能不堪设想,轻则造成经济损失,重则需要付出生命代价.为此,这类软件在正式投入使用之前必须经过彻底的测试,对其正确性加以保证,其中不可避免地要涉及对实时系统轨迹的获取以及对轨迹正确性的验证,即测试预言(test oracle).

测试预言的输入即实时系统的轨迹,为了获取此轨迹,要么在系统外部进行监测,要么在系统内部插入断言

* 本文为 2005 年中国计算机大会推荐优秀论文.Supported by the National Natural Science Foundation of China under Grant Nos.60233020, 90104007, 60303013 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2005AA113130 (国家高技术研究发展计划(863)); the Program for New Century Excellent Talents in University (新世纪优秀人才支持计划)

Received 2005-06-15; Accepted 2005-12-16

并输出.两种方法各有利弊,外部监测不会打扰系统运行但可观测的内容受限,而断言的插入则会打扰实时系统的实际运行,进而可能改变系统的时间特性.在测试预言的相关研究中,主要精力放在如何从实时规约自动生成可运行的预言,而对于获取实时系统轨迹则甚少讨论.

本文提出一种折衷的方法,根据规约的内容选择获取轨迹的方法,并使用最差情况执行时间(worst-case execution time,简称 WCET)分析技术,对插桩后的系统是否能够在一定时间容忍度下正常运行进行计算分析,并对由于断言的插入而导致对系统时间的影响提出补偿方法.

本文第 1 节对测试预言与 WCET 分析技术作简介.第 2 节提出基于规约内容和系统特性的轨迹获取方法.第 3 节利用 WCET 技术计算插桩断言对实时系统带来的打扰,并提出相应的补偿措施.第 4 节介绍相关工作及结论.

1 测试预言与 WECT 分析技术

1.1 测试预言

测试预言是一种检验待测系统在特定执行下是否正确运行的方法[1],由两部分构成:一部分是预言信息,它描述程序行为是否正确的判据;另一部分是监控过程,它根据相应的预言信息来检验程序实际执行结果的正确性.实时系统通常具有复杂的反应式和实时特性,它们通常会涉及到非常复杂的事件时序关系和比较精确的时间约束,需要自动化方法和工具的支持,在确保测试结果正确性的同时提高测试效率.实时系统的轨迹提取是测试预言得以顺利运行的重要前提.

1.2 WCET分析

为了保证实时系统中每个任务都能在其时限范围内完成,有必要分析每个任务在最差情况下的执行时间(WCET).WCET 分析是实时系统的一个重要研究领域[2],它计算给定应用程序代码片断的处理器执行时间的上限.WCET 分析保证分析的安全性(safety)和精确性(tightness),安全性是指不能低估最差执行时间;精确性是指必须提供可接受的高估值.

面向 C 源程序的 WCET 分析过程如图 1 所示.词法和语法分析程序根据 C 语言的语法将 C 源程序翻译为内部表示的中间代码,这些中间代码包含了程序的结构信息.程序流分析程序根据程序结构信息获得针对 WCET 分析的程序流信息,其中包括函数的调用关系、递归调用是否存在、循环的界限、不同 if 语句之间的依赖关系、if 语句的分支是否包括循环终止语句(break,return),等等.

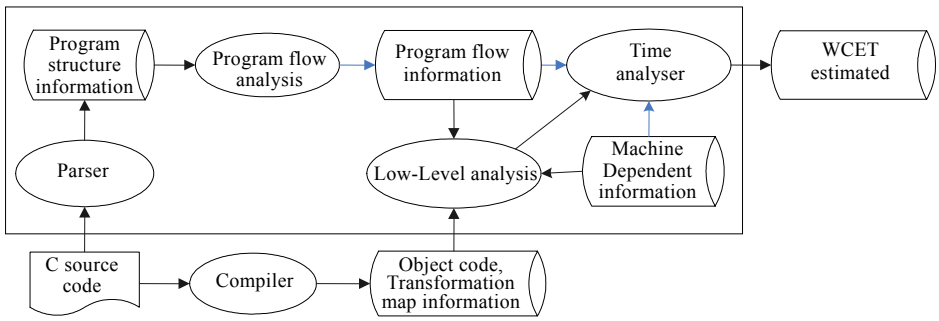

Fig.1 The WCET analysis process for C source code

图 1 基于 C 源程序的 WCET 分析过程

C 语言编译器将 C 源程序翻译成目标代码,并获得从源程序到目标代码的映射信息.低层分析程序利用这些信息以及依赖于机器的信息和程序流信息,给出每一条指令或者基本块[3]的时间行为.其中,依赖于机器的信息包括各类指令的时间特征以及系统的配置信息.根据程序流信息和基本块的时间特征,时间分析程序即可计算出指定程序代码段的处理器执行时间上限,即所需的任务或者程序代码段的 WCET 预测值.

2 实时系统的轨迹获取

测试预言主要用于监控(monitoring)待测系统的任务执行,判断其是否满足某些约束条件.F. Jahanian[4]曾经将监控方式分为同步和异步两种.同步监控即程序员在待测系统的特定位置显式地插入某些语法成份,通过直接操作及相关任务共享的事件历史来判断待测系统行为是否满足约束条件,测试预言的运行与待测系统的运行是同步进行的,如图 2 所示.这种方式虽然无须考虑获取系统运行状态后的通信开销问题,但是,由于测试预言的运算通常计算量相对较大,因此,测试预言本身的运行会对待测系统的运行环境产生较大影响,从而会极大地影响待测系统的实际运行结果.

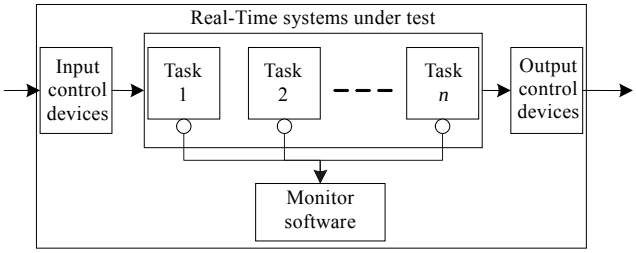

Fig.2 Synchronous monitoring method

图 2 同步监控方式

异步监控方式则将获取待测系统产生的事件与对事件是否满足约束的判断相分离,各自异步执行,互不干扰,对待测系统运行序列是否与约束相冲突单独进行检测并进行相应处理,如图 3 所示.这种方式对于实时系统测试而言显然是比较适合的方式,也是本文主要考虑的监控方式.监控程序可分为两部分:一部分为监控客户方,放入待测实时系统,作为实时系统的一个任务参与调度;另一部分为监控软件,在被测实时系统之外独立运行.

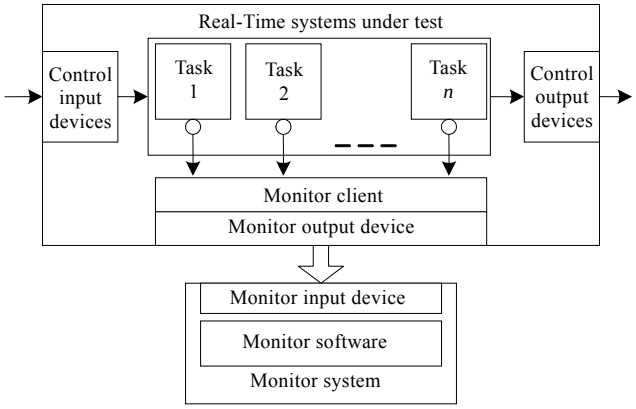

Fig.3 Asynchronous monitoring method

图 3 异步监控方式

此外,监控还可分为外部监控与断言监控两种.外部监控通常采用硬件监控[5,6],通过探听和匹配目标系统的总线信号检测事件的出现.硬件监视对于待测的实时系统不会产生任何打扰,但难以监控到系统所有状态变化,特别是当系统约束中出现外部不可见的状态时,外部监控通常是无能为力的.

相对于外部监控而言,断言监控则在待测系统的源程序内通过插入断言而获取系统的相关事件和状态.这种方式当然能够获取系统任何相关的状态和事件,但由于时间可预测性是实时系统的基本要求之一,因此,在使用断言监控方式获取系统状态时,必须量化由于断言插入而对实时系统引起的时间干扰.

基于上述各种监控方式的特点,本文提出基于规约内容和系统特性的轨迹获取方法.

2.1 基于规约内容和系统特性的轨迹获取

所谓基于规约内容和系统特性的轨迹获取方法,是指根据规约中所涉及的事件类型,即是外部事件还是内部状态,以及系统事件和状态变迁特性,即是突发性还是周期性,以决定获取系统轨迹的方式.

我们采用异步监控方式,将监控程序置于待测实时系统外部.若规约中仅涉及到系统的外部事件,则采取外部监控方式;若规约中既涉及到外部事件又涉及到内部状态,此时,我们采取混合获取方式,如图 4 所示,仅在系统内部状态可能发生改变的代码后插入断言,若断言发现系统内部状态发生了改变,则输出该内部状态值及状态发生改变的时间;而对于规约中涉及到的外部状态,则仍采用外部监控方式,以高速轮询获取外部事件发生的时间.在这种方式下,监控程序的客户方所花费的 CPU 时间会很少,进而能够使实时系统的原调度策略,在包括该客户方程序的前提下,仍然能够完成插入断言后的实时任务的调度.而监控程序对实时系统运行是否满足规约的判断处理功能[7]则放在独立的外部监控软件中.这种混合获取方式,一方面能够获取规约所需的所有系统事件和状态;另一方面又能够尽可能地减小了由于断言的插入而导致对系统时间的影响,同时具备了外部监控和断言监控两者的优点.

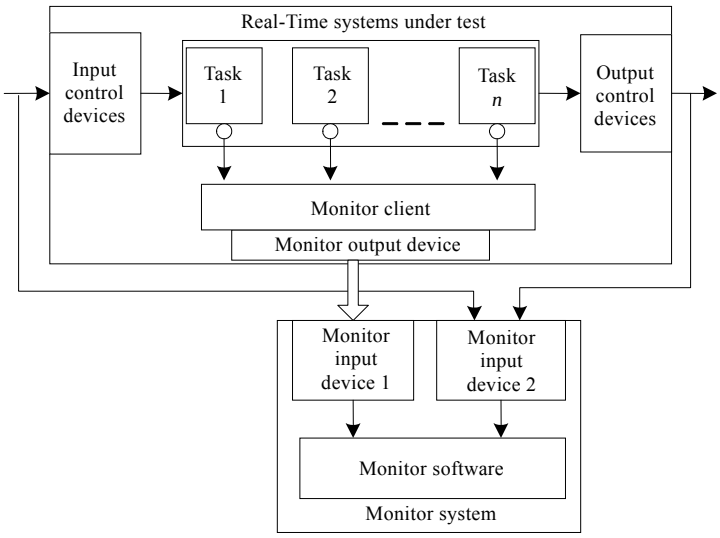

Fig.4 Mixed monitoring method

图 4 混合监控方式

此外,插入断言后对实时系统的影响必须加以分析.断言在系统状态可能发生改变的部分被插入,当系统状态发生改变时,断言向系统外部的监控程序发送当前状态值及状态发生改变的时间.在这里,断言向外部监控程序的通信开销是对实时系统的时间行为有较大影响的主要因素之一.如果断言的每次执行都要和外部设备发生通信,那么与外部设备的每次通信皆会导致 1 个或者多个相应设备服务程序(比如设备驱动程序)的运行,而这些服务程序往往处于很高的优先级(如在 VMS 操作系统中,设备驱动程序处于核心的设备优先级中,设备优先级比应用级的所有任务的优先级都高),并且这些服务程序的执行时机往往发生在数据传输前后,因而经常会打断实时应用任务的执行.当把这些服务程序当作任务进行调度时,会使调度策略变得非常复杂.不仅如此,由于 WCET 是按照不被中断估算出来的,因此,对于具有类似高速缓存这样具有全局特性的处理器而言,通信的大量存在还将使所有应用任务的执行时间难以确定.为此,我们进一步提出一种折衷方法,即根据系统特性决定获取系统轨迹方式.实时系统的运行应当是时间可预测的,因此,设计人员对于系统运行特性应当比较清楚.基于这种认识,我们在进行测试时,可以根据系统运行的特性选择监控客户方与监控软件之间的通信方式.由于中断对实时系统的打扰较大,我们应当尽可能地减少断言引起的通信中断.为此,我们提出批处理式的通信.如果实时系统的状态变迁具有突发性和集中性,亦即经常在极短时间内出现状态变迁尖峰,那么对通信的批处理则主要

考虑使用定量发送方式,即监控客户方对要发送至监控软件的内部状态进行排队,当状态变迁达到一定数目时才向监控软件发送.如果实时系统的状态变迁具有周期性和均匀性,亦即状态变迁的频率趋于一定,则考虑使用定时发送方式,由监控客户方定时地向监控软件发送内部状态变迁信息.批量发送状态变迁信息的优势在于充分利用实时系统的运行特性,尽量减少监控客户方发送中断的次数,并且可以通过适当调整监控客户方发送中断的优先级来减小对实时系统实际运行的影响.当然,由于内部状态的获取采用了批处理的发送方式,因而监控软件在收到批量状态变迁信息后,应当有一个与外部事件发生信息按时间顺序重新组合的过程.在此,我们不再详细讨论.

2.2 基于规约内容和系统特性的轨迹获取方式对实时系统时间行为的影响

断言的插入将会对实时系统的实际运行产生影响.因此,实时系统在设计和编码时应当为测试留出一定的时间余量,以支持实时测试.假定监控客户方对应于两个任务:发送开始和发送完成,并且假定这两个任务对其他实时任务的实时性没有影响.此时,原实时系统应当为测试预留的时间余量为

$$TimeForTest = WCET(TestClient) + \sum_{i \in Tasks} (WCET_{Instr}(i) - WCET_{Uninstr}(i)) \quad (1)$$

其中:$Tasks$ 为实时系统的任务集合;$WCET(TestClient)$为监控客户任务的 WCET;$WCET_{Instr}(i)$和 $WCET_{Uninstr}(i)$分别为第 i 个任务被插装之后和之前的 WCET.

除了系统预留时间余量以外,断言对原实时系统的插装有可能引起实时系统的时序行为发生改变.例如,规约要求事件 B 在事件 A 发生后的 1 个单位时间内发生,即$\Box(A \rightarrow \Diamond_{[0,1]}B)$,若在原实时系统的某次运行中,事件 A 发生在时刻 10,事件 B 发生在时刻 11,则此时系统满足规约.但为了测试事件的顺序,分别在事件 A,B 之后插入断言 D_A和 D_B,再次运行实时系统时,事件 A 发生在时刻 13,事件 B 发生在时刻 15,此时系统不再满足规约.

为了保证获取的实时系统的时序行为不会因断言的插入而受到影响,需要将断言引入的时间偏移消除,从而消除断言对原实时任务的影响.插入断言对原实时任务的影响包括两个方面:一是插入断言本身的执行时间影响了原实时任务的指令执行时刻;一是断言的插入和执行影响了原实时任务的指令执行时间,例如,改变了原实时任务的指令地址或者断言的执行指令与原实时任务的指令地址发出冲突从而改变了缓存行为等.

我们将在下一节描述插入断言的 WCET 分析及原实时任务指令执行时刻的改变值,并在第 4 节讨论相应的时间补偿方法.

3 插装断言的时间分析和补偿

由于断言的存在,必然会对原实时系统的任务执行产生影响,而这种影响可能会导致系统行为由单独运行时的正确时序变得不再正确,即会使实时系统的事件发生和状态变化时间产生偏移.为了消除这种偏移,需要使用 WCET 分析技术对插装后的断言执行时间及插入断言对原系统的时间影响进行分析.

3.1 插装断言对被测系统的时间影响分析

插入点 CS 是程序中需插入断言的一个位置.断言的构成是一段程序,在插入时将其作为一个完整单位插入.因此,插入点一定位于一个语句之后、另一个语句(或者是函数末尾)之前.不妨假定插入断言为 DS,原程序语句为 S_1;S_2,插入点位于语句 S_1 之后,则插装后的语句序列为 S_1;DS;S_2.

检测点 JS 是程序中的一个位置,在此位置度量插入断言对原程序语句的时间影响.检测点 JS 不可能位于插入断言之中,它通常位于插入断言之后的原程序的某个位置.假定程序的开始位置为 KS(开始位置不一定是任务的第 1 个语句[8]),插装之前的语句序列为 S_1;S_2,插装后的语句序列为 S_1;DS;S_2.我们通常定义检测点 JS 在语句 S_2 之前.

对于任一检测点 JS,断言插装之后引起的时间变化量为

$$TimeChanged(JS)=Time_{Instr}(KS \rightarrow JS)-Time_{Uninstr}(KS \rightarrow JS) \quad (2)$$

其中:$KS \rightarrow JS$ 表示从 KS 到 JS 的程序运行路径;$Time_{Instr}(KS \rightarrow JS)$和 $Time_{Uninstr}(KS \rightarrow JS)$分别为插装之后和插装之前从 KS 到 JS 的程序路径时间.

由于插装后程序语句由原程序语句和插装语句组成,因此有

$$TimeChanged(JS)=DSTime(JS)+ChangedUninstr(JS) \tag{3}$$

其中:

$$DSTime(JS)=\sum_{\substack{DS_i\in KS\to JS\\ DS_i\in DSSet}} DS_i \tag{4}$$

$$ChangedUninstr(JS)=\sum_{\substack{DS_i\in KS\to JS\\ DS_i\notin DSSet}} (Time_{Instr}(S_i)-Time(S_i)) \tag{5}$$

分别表示插装语句所花费的时间和原程序语句插装之后引起的时间变化量;*DSSet* 表示插装语句的集合.

由公式(3)可知,插装之后该点的时间变化 *TimeChanged*(*JS*)包括两部分:一部分是插装语句所花费的时间 *DSTime*(*JS*);另一部分是原程序语句插装之后引起的时间变化量,后者与前者有关.同理,*TimeChanged*(*JS*)的计算精度也由这两部分的计算精度来决定.

对于简单的CISC处理器,两条指令的执行时间等于每条指令执行时间的简单累加,插入断言对原程序语句的执行时间没有影响,因此有 *ChangedUninstr*(*JS*)=0;但对于现代处理器则不然.

Engblom[9]把现代处理器特征对时间特性的影响分为全局和局部两类,比如高速缓存和分支预测是全局的,而流水线是局部的.插入断言对原程序语句时间行为的影响既包括局部特征也包括全局特征.例如,对于插装后的语句序列 S_1;*DS*;S_2,由于 *DS* 的插入,可能使得 S_2 所使用的流水信息发生变化[10],从而使得 S_2 的执行时间发生改变.同样是由于 *DS* 的插入,有可能使得 S_2 对应的内层地址发生改变,从而有可能使 S_1 和 S_2 的缓存特征发生改变,进而改变 S_1 和 S_2 的时间行为.

这里主要分析插入断言的时间行为.因此假定对于检测点 *JS*,*TimeChanged*(*JS*)总能够精确计算.需要说明的是,由于我们这里有监控程序,能够精确获取程序的执行轨迹,所以对于简单的 CISC 处理器,或者一般的具有流水线和高速缓存的现代处理器,能够精确地计算 *TimeChanged*(*JS*).

3.2 插装断言的构造

在对实时系统源程序进行断言插装时,我们需要在所有实时规约所关心的内部状态可能发生变化的语句之后插入断言,并且为了判断状态是否发生变化,需要在相关状态初始化阶段获取状态初值.为此,断言的结构在这两种情况下可分为 4 种类型,即初始化断言、条件断言、全称断言和存在断言,见表 1.

Table 1 Four types of inserted assertion

表 **1** 插入断言的 4 种类型

	Conditional	For all	Exist	Initial
Meaning	for judging the change of state	for the formulas contains more than one variable		Others
Statement structure	if (*E*){S_{t1};…;S_{tn}};	for (*i*){if (*E*){S_{t1};…;S_{ti};break;}S_{o1};…;S_{om}}		S_1;S_2;…;S_n
Examples	Fig.5	Fig.7	Fig.8	Fig.6, Fig.9, Fig.10

在图 5~图 10 中,*EventValue* 为某变量的当前值;*OldValue* 为该变量的前次值;*t* 为该变量发生变化的时间.*Send*()将该变量的当前值与值发生变化的时间发送给监控客户方.

举例来说,规约$\square_{[0,100]}(\forall_{i\in 1..20}(A[i]!=Emergency))$的意思是在 100 单位时间内,数组 *A* 的任意分量皆不能处于 *Emergency* 状态.而规约$\square_{[0,100]}(\exists_{i\in 1..20}(A[i]==Emergency))$的意思则是在 100 单位时间内,存在数组 *A* 的某分量处于 *Emergency* 状态.

在图 7 中,*EventValue*[*i*]表示命题 *A*[*i*]!=*Emergency* 的真值;在图 8 中,*EventValue*[*i*]表示命题 *A*[*i*]==*Emergency* 的真值,而 *EventValue* 则表示$\forall_{i\in 1..20}(A[i]!=Emergency)$和$\exists_{i\in 1..20}(A[i]==Emergency)$的真值.

3.3 插装断言的时间计算

由表 1 可知,插装断言有 3 种形态,我们可以按照这 3 种形式讨论相应的时间计算:

1. 顺序语句序列 S_1;S_2;…;S_n

对于简单的 CISC 处理器,其执行时间为

$$Time(S_1;S_2;\ldots;S_n)=Time(S_1)+Time(S_2)+\ldots+Time(S_n) \tag{6}$$

对于流水线处理器,其执行时间的计算通常使用保留表[11],即 n 条指令 $S_1;S_2;\ldots;S_n$ 的执行时间为从 S_1 到 S_n 的保留表串联之后,从第 1 条指令第 1 个阶段到最后一条指令的最后一个阶段之间的周期(cycle)数.

当具有高速缓存时,对于缓存丢失(miss)的情况,需要把丢失的时间补上.当既有流水线又有高速缓存时,也可以有更精确的计算[12].

2. 条件语句 if (E) then S_1; else S_2;

条件语句相当于不同条件下的顺序语句,即

$$Time\ (\text{if } (E) \text{ then } S_1;\ \text{else } S_2;):=\begin{cases} Time(E;S_1), & \text{if } E=\text{TRUE} \\ Time(E;S_2), & \text{if } E=\text{TRUE} \end{cases} \tag{7}$$

这里,E 为条件表达式代码;$Time(E;S_1;)$为执行条件表达式之后紧接着执行语句 S_1 的执行时间;$Time(E;S_2;)$类似.

3. 循环 for (i){if (E){$S_{t1};\ldots;S_{tj}$;break;}$S_{o1};\ldots;S_{om}$}

假定循环迭代次数为 $n>0$,则有

$Time$(for (i){if (E){$S_{t1};\ldots;S_{tj}$;break;}$S_{o1};\ldots;S_{om}$})

$$Time(\text{for } (i)\{\text{if } (E)\{S_{t1};\ldots;S_{tj};\text{break};\}S_{o1};\ldots;S_{om}\}) = \begin{cases} Time(E;S_{o1};\ldots;S_{om})*(n-1)+Time(E;S_{t1};\ldots;S_{tj}), & \text{if } E \text{ evaluate to TRUE once} \\ Time(E;S_{o1};\ldots;S_{om})*n, & \text{if } E \text{ evaluate to TRUE} \end{cases} \tag{8}$$

从上述公式能够看出:公式(7)的计算要求能够确定条件表达式 E 的计值结果;而公式(8)的计算不仅要求条件表达式 E 的计值结果,还要求能够确定循环的迭代次数.这些信息通常是能够获得的,比如通过对断言插入适当标识,监控程序就能够确定相应的执行轨迹,从而确定插装断言的执行路径.

3.4 目标系统的时间计算和修正

对于检测点 JS,断言插装之后引起的时间变化量 $TimeChanged(JS)$不仅与插入的断言有关,还与目标系统的原程序本身有关.前面我们假定 $TimeChanged(JS)$总能够精确计算.要精确计算程序的执行时间,首先需要考虑程序的循环.这里仅讨论对循环的处理,对其他语法成分的处理,可以参见文献[11−13].

与通常对循环迭代次数的估算[14,15]不同,由于有监控程序的存在,并且 $TimeChanged(JS)$的计算不要求在事先获得,因此,通过诸如在循环中插入标识之类的方法,监控程序能够及时获知循环的迭代次数.下面假定已知循环处于第 $i>0$ 次迭代,这里,$Time(KS\rightarrow JS)$表示 $Time_{Instr}(KS\rightarrow JS)$和 $Time_{Uninstr}(KS\rightarrow JS)$.

注意到,KS 位于循环的最外层.当 JS 和 KS 位于同一个循环层次时,很显然,$Time(KS\rightarrow JS)=Time(P_{KS\rightarrow JS})$,$P_{KS\rightarrow JS}$ 为从 KS 到 JS 的程序语句序列.

假定 JS 位于比 KS 更深的一个循环层次,即 JS 位于循环 for (E) S 之中,则有

$$Time_i(KS\rightarrow JS)=Time(P_{KS\rightarrow WKS})+(i-1)\times Time(E;S)+Time(E;P_{WKS\rightarrow JS}) \tag{9}$$

其中:WKS 为语句 S 的开始位置;$P_{KS\rightarrow WKS}$ 为从 KS 到 WKS 的程序语句序列;$P_{WKS\rightarrow JS}$ 为从 WKS 到 JS 的程序语句序列.

监控程序能够根据断言中的分支判断取值和循环的迭代次数,利用 WCET 分析技术,根据公式(6)~公式(8)能够确定 $Time(P_{KS\rightarrow WKS})$,$Time(E;S)$和 $Time(E;P_{WKS\rightarrow JS})$.

从公式(9)我们还可以得到:

$$Time_i(KS\rightarrow JS)=Time_{i-1}(KS\rightarrow JS)+Time(E;P_{WKS\rightarrow JS}) \tag{10}$$

可以用同样的方法处理其他形式的循环和多重循环.

断言的插入使得事件的发生时间发生了改变,从而使得事件之间的实时关系有可能发生改变.通过上述的时间计算,我们可以对断言的观测时间进行修正,使用修正时间可以减少或者消除插入断言对实时关系的影响.

假定监控程序对断言 A 的观测时间为 T_A,修正后的时间 $T_{A'}$ 为

$$T_{A'}=T_A-TimeChanged(JS) \tag{11}$$

4 相关工作及结论

传统的硬件监控方法使用特殊的硬件(如指定的处理器)通过对待测系统总线的侦听和匹配来获取事件的发生.这些方法对系统的监控是非入侵的,但无法监控到所有事件.P.S. Dodd,J. Joyce 和 H. Tokuda 等人提出的软件监控方法[16,17]在软件内部插入软探针来获取事件,尽管这些方法可以获得系统的所有事件,但软件探针为待测系统所带来的影响却没有深入讨论.

F. Jahanian 等人[7]提出将软插件(类似断言)作为带时间约束的任务对待测系统的行为进行监控.该任务与实时系统一起进行调度,从而能够量化由于软件插件而引起的实时系统行为偏差.

Peters 和 Parnas[18]提出一种硬件监控与软件监控相结合的监控方法,硬件监控使用特殊的输入设备对待测系统进行观察.这一工作与我们的工作有许多类似之处,但没有对软件监控引起的系统行为偏差进行讨论和计算.

针对实时系统的测试预言,本文讨论了不同的监控方式对目标实时系统时间行为的影响,进而提出了一种介于软件和硬件之间的折衷的混合监控方式,它对目标系统的时间干扰比软件方式要小.为了完全测试目标系统,必须对目标系统插装断言,断言的插装不可避免地会对目标系统的时间行为产生影响.为了尽可能地消除插入断言对目标系统的影响,本文还提出一种基于 WCET 分析的目标系统时间补偿方法.该方法能够精确地计算插入断言对目标系统的影响,从而有效地消除插入断言对目标系统时间行为的影响.

References:

[1] Baresi L, Young M. Test oracles. Technical Report, CIS-TR01-02. University of Oregon, 2001. http://www.cs.uoregon.edu/-michal/pubs/oracles.html

[2] Puschner P, Burns A. Guest editorial: A review of worst-case execution-time analysis. Real-Time Systems, 2000,18(2-3):115−128.

[3] Aho AV, Sethi R, Ullman JD. Compilers, Principles, Techniques, and Tools. New York: Addison-Wesley, 1997.

[4] Jahanian F, Rajkumar R, Raju SCV. Runtime monitoring of timing constraints in distributed real-time systems. Real-Time Systems, 1994,7(3):247−273.

[5] Liu AC, Parthasarathi R. Hardware monitoring of a multiprocessor system. IEEE Micro, 1989,9(5):44−51.

[6] Tsai JJP, Fang KY, Chen HY. A noninvasive architecture to monitor real-time distributed systems. Computer, 1990,23(3):11−23.

[7] Jahanian F. Run-Time Monitoring of Real-Time Systems in Advances in Real-time Systems. Prentice-Hall, 1995. 429−454.

[8] Hu EYS, Bernat G, Wellings A. Addressing dynamic dispatching issues in WCET analysis for object-oriented hard real-time systems. In: Proc. of the 5th IEEE Int'l Symp. on Object-Oriented Real-Time Distributed Computing. Los Alamitos: IEEE Computer Society Press, 2002. 109−116.

[9] Engblom J. Processor pipelines and static worst-case execution time analysis [Ph.D. Thesis]. Uppsala: Acta Universitatis Upsaliensis, 2002.

[10] Engblom J, Jonsson B. Processor pipelines and their properties for static WCET analysis. In: Sangiovanni-Vincentelli AL, Sifakis J, eds. Proc. of the 2nd Embedded Software Conf. LNCS 2491, Grenoble: Springer-Verlag, 2002. 334−348.

[11] Shaw AC. Reasoning about time in higher level language software. IEEE Trans. on Software Engineering, 1989,15(7):875−889.

[12] Healy CA, Arnold RD, Mueller F, Whalley D, Harmon MG. Bounding pipeline and instruction cache performance. IEEE Trans. on Computers, 1999,48(1):53−70.

[13] Lim SS, Bae YH, Jang GT, Rhee BD, Min SL, Park CY, Shin H, Park K, Moon SM, Kim CS. An accurate worst case timing analysis for RISC processors. IEEE Trans. on Software Engineering, 1995,21(7):593−604.

[14] Healy CA, Sjödin M, Whalley DB. Bounding loop iterations for timing analysis. In: Proc. of the 4th IEEE Real-Time Technology and Applications Symp. (RTAS). Denver, 1998. 12−21.

[15] Gustafsson J, Ermedahl A. Automatic derivation of path and loop annotations in object-oriented real-time programs. Parallel and Distributed Computing Practices, 1998,1(2):61−74.

[16] Dodd PS, Ravishankar CV. Monitoring and debugging distributed real-time programs. Software-Practice and Experience, 1992, 22(10):863−877.

[17] Joyce J, Lomow G, Slind K, Unger B. Monitoring distributed systems. ACM Trans. on Computer Systems, 1987,5(2):121−150.

[18] Peters DK, Parnas DL. Requirements-Based monitors for real-time systems. IEEE Trans. on Software Engineering, 2002,28(2): 146−158.

王馨(1976－),女,山东济南人,博士,主要研究领域为软件测试,实时系统.

王戟(1969－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为高可信软件技术,Agent 软件方法学.

姬孟洛(1963－),男,博士生,主要研究领域为实时系统分析,面向对象设计.

齐治昌(1942－),男,教授,博士生导师,主要研究领域为软件工程,软构件开发与验证.

2006 年全国开放式分布与并行计算学术会议

征 文 通 知

由中国计算机学会开放系统专业委员会主办、西北大学信息学院（计算机系）承办、陕西省计算机学会协办的“2006 年全国开放式分布与并行计算学术会议”将于 2006 年 10 月 19－21 日在西安召开，现将会议征文的有关事项通知如下：

一、征文范围(包括但不限于下列方面)

1. 开放式分布与并行计算模型、体系结构、算法及应用；
2. 下一代开放式网络、数据通信、网络与信息安全、业务管理技术；
3. 开放式海量数据存储与 Internet 索引技术，分布与并行数据库及数据/Web 挖掘技术；
4. 开放式机群计算、网格计算、Web 服务、P2P 网络及中间件技术；
5. 开放式移动计算、移动代理、传感器网络与自组网技术；
6. 分布式人工智能、多代理与决策支持技术；
7. 分布、并行编程环境和工具；
8. 分布与并行计算算法及其在科学与工程中的应用；
9. 开放式虚拟现实技术与分布式仿真；
10. 开放式多媒体技术与流媒体服务，包括媒体压缩、内容分送、缓存代理、服务发现与管理技术。

二、征文要求

论文必须是未正式发表的、或者未正式等待刊发的研究成果。论文格式仿照《微电子学与计算机》刊物的格式，应包括题目、作者、所属单位、摘要、关键词、正文和参考文献。论文中、英文均可，一般不超过 5000 字，一律用 Word2002 格式排版，提供 A4 激光打印稿一式两份，并将论文电子版上传到会议网站上或发送 Email 至 chenxr@mail.xjtu.edu.cn。经程序委员会审查合格的论文，将全部在中国计算机学会会刊《微电子学与计算机》（核心期刊）正刊上发表。会议将评选大会优秀论文，予以奖励并推荐到一级学报发表。

三、重要日期

会议时间：2006 年 10 月 19～21 日　　截稿日期：2006 年 7 月 15 日　　录用通知：2006 年 7 月 30 日

四、联系方式

投稿地址：西安西北大学信息学院　陈晓江 收(请在信封上注明 DPCS2006)　　邮政编码：710069
联系电话：029-88308273（房鼎益、陈晓江）
电子邮件：chenxr@mail.xjtu.edu.cn (请在邮件主题中注明 DPCS2006)
会议主页：http://disnet.nwu.edu.cn/DPCS2006 或 http://cs.nju.edu.cn/dpcs
专委会联系人：南京大学计算机系陈贵海电话：025-58916715, 电邮：gchen@nju.edu.cn