Journal of Computer Applications
计算机应用,2014,34(5):1418 - 1422
ISSN 1001-9081
CODEN JYIIDU
2014-05-10
http://www.joca.cn

文章编号:1001-9081(2014)05-1418-05
doi:10.11772/j.issn.1001-9081.2014.05.1418

面向星载操作系统的控制流错误检测方法

明月伟*,宁　洪,邓胜兰
(国防科学技术大学 计算机学院,长沙 410073)
(*通信作者电子邮箱 myw123@163.com)

摘　要:空间高能粒子辐射严重影响航天计算的可靠性,必须采取有效措施对其进行加固。相比使用抗辐射器件,使用软加固的商用器件具有性能高、成本低、开发速度快等优势。然而,目前的软加固研究主要适用于应用程序,对操作系统软加固方法的研究还较少。鉴于此,提出了一种面向星载操作系统的控制流错误检测方法。该方法结合星载操作系统自身特点,着眼于线程执行,将每个线程视为一个函数调用序列,通过在函数入口和出口处插入检测语句,监测各线程的执行,实现控制流错误检测。实验结果表明,该方法能将星载操作系统的控制流错误覆盖率提高约25%。

关键词:单粒子效应;星载操作系统;软加固;控制流错误检测;函数级
中图分类号:TP316.2　　**文献标志码**:A

Control flow checking method for on-board operating system

MING Yuewei*, NING Hong, DENG Shenglan
(*College of Computer, National University of Defense Technology, Changsha Hunan 410073, China*)

Abstract: The space high-energy particle radiation has a serious influence on the reliability of the space computation. Effective radiation hardening measures must be taken to overcome this problem. Compared with the use of radiation hardening devices, using the soft reinforcement commercial devices can enjoy the advantages of high performance, low cost, fast development and so on. However, the present research on soft reinforcement is mainly suitable for the application level, while there is very little research on soft reinforcement methods for the operating system. A control flow checking method for the on-board operating system was proposed to solve this problem. Taking account of the characteristics of the on-board operating system, the proposed method regarded each thread as a sequence of function calls and monitored the thread execution through inserting test statements in the entry and exit of a function to achieve the control flow error detection. The experimental results indicate that the proposed method can increase the control flow fault coverage of the on-board operating system by 25%.

Key words: Single Event Effect (SEE); on-board operating system; soft reinforcement; control flow checking; function level

0　引言

空间高能粒子辐射严重影响航天计算的可靠性。主要因为空间高能粒子进入半导体数字集成电路后,会电离出大量电荷,导致集成电路出现逻辑翻转、锁定,甚至可能发生烧毁。这种由高能粒子引起半导体数字集成电路逻辑紊乱或失效的现象称为单粒子效应(Single Event Effect, SEE),具体可以分为单粒子翻转、单粒子闩锁及单粒子烧毁三种现象。本文主要考虑由单粒子翻转导致的硬件瞬时故障,其直观影响是可能使存储器、寄存器或其他硬件逻辑器件的某位从逻辑0变成逻辑1,或者反之[1],进而导致标志错误、程序走飞、死锁或者计算错误等后果,影响系统的正常运行。

通常情况下,我们采用经过硬件特殊加固的抗辐射器件抵御空间恶劣环境。这类产品能有效克服空间辐射的影响,但是在性能、价格、功耗、体积、研发周期等方面相比现今的商用现货(Commercial Off The Shelf, COTS)产品都存在很大的差距。因而使用COTS器件代替抗辐射加固器件构建航天计算平台成为当下研究热点。这样不仅满足了日趋复杂的空间科学任务对高性能的需求,而且具有成本低、应用灵活、开发效率高等优势。但是COTS器件在空间辐射条件下不能保证可靠性,所以就需要使用软加固技术对其进行可靠性增强。软加固技术指的是以软件手段容硬件之错,通过在软件层次实现错误检测和恢复方法,提高系统可靠性。

目前的软加固技术研究主要适用于应用程序层次。一般从控制流和数据流两方面着手,对应用程序进行加固。这方面已取得不少卓有成效的研究成果。Stanford CRC实验室提出了软件实现的硬件故障容忍(Software Implemented Hardware Fault Tolerance, SIHFT)技术,主要包括CFCSS(Control-Flow Checking by Software Signatures)[2]、EDDI(Error Detection by Duplicated Instructions)[3]、ED^{4}I(Error Detection by Diverse Data and Duplicated Instructions)等;美国普林斯顿大学则提出了以SWIFT(Software Implemented Fault Tolerance)为代表的软加固方法。此外,欧洲TIMA实验室、法国LAAS-CNRS实验室、瑞典Chalmers大学、意大利都灵理

收稿日期:2013-10-08;**修回日期**:2013-12-02。
作者简介:明月伟(1988 -),男,山东邹平人,硕士研究生,CCF会员,主要研究方向:系统软件、软件容错;　宁洪(1961 -),女,湖南衡阳人,教授,主要研究方向:数据集成与分析、星上计算;　邓胜兰(1961 -),女,湖南衡阳人,研究员,硕士,主要研究方向:嵌入式操作系统。

《计算机应用》Journal of Computer Applications

工大学等研究机构在应用层次瞬时故障检测方面,也开展了一系列软件抗辐射技术的研究和实验,取得了积极的成果。

然而这些研究主要集中在应用层次,难以在操作系统加固中直接运用。在针对操作系统的加固研究方面,Nicolescu 等以 MicroC/OS-Ⅱ为例研究了嵌入式实时操作系统(Real-Time Operating System, RTOS)对于瞬时故障的敏感性,总结了瞬时故障对 MicroC/OS-Ⅱ所造成的各种错误类型及所占比例[4-5]。Neishaburi 等提出将 RTOS 的部分功能(如任务调度功能)使用硬件实现,然后通过加固硬件达到对 RTOS 进行加固的目的[6]。

尽管操作系统加固方面已有上述一些成果,但是对操作系统这种复杂系统软件如何进行软加固的研究还较少。而愈加复杂的空间科学任务决定了未来的航天计算中星载操作系统必将会被广泛使用,且作用至关重要,因此有必要对其软加固方法进行探索。针对这一问题,本文提出了一种面向星载操作系统的控制流错误检测方法。该方法通过监测各线程的函数执行序列,实现控制流错误检测。最后,在一个具体的嵌入式实时操作系统 SYS/BIOS 上进行了具体实现。

1　星载操作系统特点

本文提到的星载操作系统是指卫星载荷部分所使用的操作系统,主要为嵌入式实时操作系统。星载操作系统内核主要由中断管理、时间管理、任务管理、任务间同步与通信、内存管理等部分组成。

星载操作系统能够支持多线程,使得程序开发更加容易,便于维护,同时能够提高系统的稳定性和可靠性。相比传统的桌面操作系统,星载操作系统更加精简,采用基于优先级抢占的调度方式,功能可裁剪,可靠性、成本、体积、功耗等都有严格要求[7]。

星载操作系统中的线程往往分为几类,包括硬中断、任务等。星载操作系统不单独运行,而是和应用程序一起编译连接成一个可执行程序。从一定程度上讲,星载操作系统可视为供应用程序调用的函数库。各线程的执行可看作对其功能函数的调用序列。

下面以 SYS/BIOS 为例具体介绍星载操作系统特点。由于星载操作系统的功能相似性,以下这些内容不仅局限于 SYS/BIOS,也适于大多数星载操作系统。

SYS/BIOS 是一个由德州仪器(Texas Instruments, TI)公司开发的可运行于 DSP、ARM 等处理器上的嵌入式实时内核,与 TI 开发工具链紧密结合、尺寸可伸缩,功能完善,易于使用。

SYS/BIOS 可以实现硬中断(Hardware interrupt, Hwi)处理机制,软中断(Software interrupt, Swi)处理机制,任务管理(Task),时间管理(Timer、Clock),任务间同步与通信(信号量、事件、邮箱),内存管理等。

SYS/BIOS 程序中主要有四种类型的调度单位(线程):硬中断线程(Hwi)、软中断线程(Swi)、任务线程(Task)、空闲线程(Idle)[8]。四种线程的优先级由低到高分别为 Idle、Task、Swi 和 Hwi,如图 1 所示。

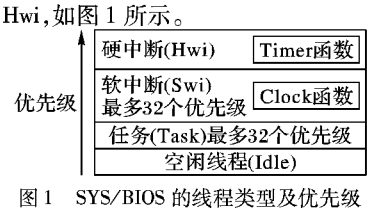

图 1　SYS/BIOS 的线程类型及优先级

1)硬中断线程(Hwi)。

Hwi 线程(也称为中断服务程序或 ISRs)是 SYS/BIOS 应用中优先级最高的线程。Hwi 线程用于执行时间要求严苛的硬实时任务。它们由发生在实时环境中的外部异步事件(中断)触发。Hwi 线程总是一直运行到结束,但是在开中断的条件下可能被由其他中断触发的 Hwi 线程临时抢占。Timer 函数运行在 Hwi 线程级。

2)软中断线程(Swi)。

软中断利用硬中断(Hwi)的概念,用软件方式进行模拟,实现宏观上的异步执行效果。软中断线程在 Hwi 线程和任务线程之间提供了另外的优先级。与 Hwi 线程由硬中断触发不同,Swi 线程通过在程序中调用某些 Swi 模块提供的 API 来触发。Swi 用于处理那些截止时间没有硬件 ISRs 严苛,但时间限制又使得它们不能作为任务运行的软实时线程。Swi 允许 Hwi 将一些非关键处理在低优先级线程上延迟执行,这样可以减少 CPU 在中断服务程序中的时间开销,从而减少关中断的时间,提高系统的实时性。Clock 函数运行在 Swi 线程级。

3)任务线程(Task)。

任务的优先级高于空闲(Idle)线程,低于软中断。任务与软中断的不同之处在于任务可以在执行期间等待(阻塞)直到所需资源可获得。SYS/BIOS 提供了若干可用于任务间同步与通信的机制,包括信号量、事件、消息队列和邮箱等。

4)空闲线程(Idle)。

在 SYS/BIOS 中 Idle 线程优先级最低并且一直循环执行。在 main 函数最后调用 BIOS_start( )后,SYS/BIOS 为每一个模块调用启动程序,然后进入 Idle 循环。Idle 循环持续执行除非有更高优先级的线程抢占 CPU。只有没有严格截止时间的函数才应该在 Idle 线程中执行。Idle 线程实际是一个最低优先级的任务

Task 和 Swi 都最多有 32 个优先级,而各 Hwi 线程具有最高且相同的优先级,一个新触发的 Hwi1 可以立刻抢占另一个正在执行的 Hwi2,除非 Hwi2 设置屏蔽字屏蔽掉某些 Hwi。图 2 是一个多线程执行的例子。

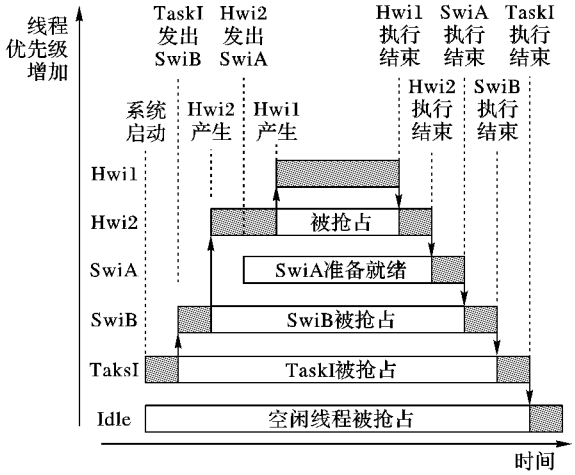

图 2　SYS/BIOS 多线程执行过程示意图

2　控制流错误检测方法

相对应用程序的软加固,星载操作系统自身的软加固有其特点。一方面相比应用程序,星载操作系统更加复杂,但功能也更固定,不像嵌入式应用那么多种多样。另一方面,星载

Journal of Computer Applications 《计算机应用》

操作系统软加固要能及时发现因单粒子效应导致的“异常”行为,又不能影响其功能特性。如果过多地加入软加固代码,势必影响星载操作系统的实时性。这两方面因素决定了对星载操作系统的软加固不能面面俱到[9],像很多应用层加固方法那样在汇编级通过编译器自动化地密集插入检查指令,而应分析星载操作系统自身特点,找到关键数据和关键路径,在保证良好错误覆盖率的前提下,从较高层次以较低频度进行控制流错误检测。

本文的控制流错误检测方法从较高抽象层次对星载操作系统进行软加固,其中线程为一级抽象,函数是低一级的抽象层次。本文将每个函数视为线程执行的基本单元,将每个线程的执行视为一个函数调用序列。利用软加固技术中控制流错误检测的经典方法——标签比对来检测控制流错误[10],这是一种纯软件方法。其基本思想是为每个基本块分配一个静态标签,在程序运行过程中,当进入一个基本块时根据当前的控制流生成动态标签,然后把静态和动态标签进行比较,不相等则说明检测到控制流错误[11-12]。

2.1 控制流错误检测方法基本思路

将每个函数视为一个基本单元,为每个函数分配静态标签。在各线程执行过程中,通过标签比对检查其函数执行序列是否正确。

1)为每个函数分配一个静态标签(signature),用以标识各函数基本单元。该标签由人为指定,要求每个函数都拥有独一无二的标签,例如可以采用四位十进制数作为一个标签,前两位表示函数所属的模块,后两位表示函数在该模块中的标号。

2)定义一个栈。每个栈元素用于存储一个函数标签。

3)在星载操作系统内核各个函数最前面加入 set(signature)语句来设置标签,set(signature)是入口事件,其将 signature 压入栈中。

4)在各函数返回(return)处之前,加入 check(signature)来进行标签比对。check(signature)是出口事件,其将栈顶元素弹出,并与 check 指定的 signature 进行匹配。如果匹配一致,则表明程序正常执行;否则,检测到控制流错误。如图 3 所示。

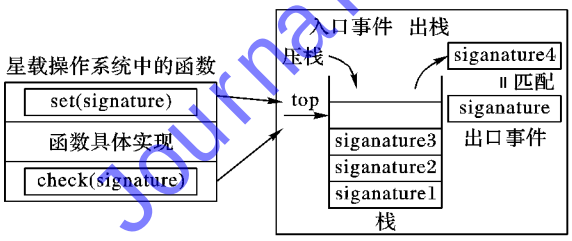

图3 控制流错误检测方法基本思路

在程序运行过程中,当进入一个函数基本块时,set(signature)将静态标签 signature 压入栈中,实质是生成了该函数的动态标签,即栈中存储的标签为动态标签;当控制流从函数基本块退出时,check(signature)将栈中弹出的动态标签与静态标签 signature 比较来检测控制流错误。

当函数调用不发生嵌套时,栈中的标签被压入,然后再被弹出匹配;而当函数调用发生嵌套时,栈会根据嵌套调用顺序依次记录各函数的标签,每当一个函数执行结束返回上一层时,就从栈顶弹出该函数的标签进行匹配。

如果在程序执行时发生控制流错误,如图 4 所示,在函数 B 执行 check(signature)时,栈顶弹出的是函数 A 的标签,从而造成不匹配,检测到控制流错误。

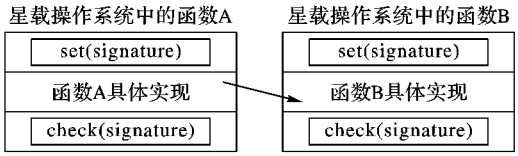

图4 控制流错误示意图

2.2 控制流错误检测方法完善

2.1 节中的方法存在一个问题。如第 1 章所述,星载操作系统有多种类型的线程。每个线程在执行过程中都会有一个函数调用序列。若各线程共用一个标签存储栈,那么各线程间的控制流标签相互影响,最终导致即使在控制流正确的情况下也会检测到标签不匹配。

举例说明这一问题。假设有两个任务 A、B,A 的优先级高于 B。A 首先执行,在执行过程中,A 调用 Task_sleep( )函数,睡眠一定时间。此时栈顶存储着 Task_sleep( )函数的标签值。紧接着 B 会开始执行,栈中又压入 B 线程函数序列中的标签值。当 A 的睡眠时间到后,A 会抢占处理器资源(因为其优先级更高),而 B 则进入阻塞态。如此一来,A 从 Task_sleep( )返回,用 Task_sleep( )的静态标签去匹配栈顶元素。而这时从栈中弹出的是 B 线程函数调用序列中的标签,从而产生标签匹配混乱,造成程序正常执行时,也会误检到控制流错误的状况。所以必须对算法进行修改完善。

本文的完善方法是让各个线程使用独立的栈,各自保存自己的函数调用序列,保证互不影响。

硬中断和软中断在不被更高优先级线程抢占时会一直运行到结束,不会自行阻塞。尽管有中断嵌套,但最外层中断在不运行结束前是不会返回的,这两种类型线程的特点决定了我们可以只使用一个栈来存储 Hwi 和 Swi 各个线程的标签。

任务是最重量级的线程,它的操作也最复杂。任务可以在执行期间等待(阻塞)直到所需资源可获得,并且任务间有复杂的同步与通信机制。如前面的例子所述,两个任务线程不能使用同一个标签存储栈。为此本文为每一个任务建立一个标签存储栈。

在 2.1 节所描述算法的基础上,再建立一个栈顶数组。该数组用于存储所建立各个栈的栈顶指针。对于任务而言,在任务控制块中加入任务 ID 字段,栈顶数组使用任务 ID 进行索引(Index)。而硬中断和软中断的各个线程使用同一个栈,其栈顶也存在栈顶数组中。如图 5 下半部分所示。

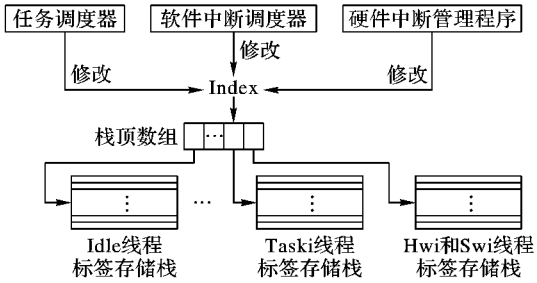

图5 控制流错误检测方法完善示意图

何时进行栈的切换是一个需解决好的关键问题。本文从线程级着手。处理器资源每次会被且仅被一个线程占用,线

《计算机应用》Journal of Computer Applications

程的切换只会发生在各个调度器中,包括任务调度器,软中断调度器和硬中断管理程序。

对任务调度器、软中断调度器和硬中断管理程序做修改,在进行线程切换之前,将即将要运行线程所对应的 ID 赋值给栈顶数组索引 Index。栈顶数组根据 Index,取出该线程所对应的栈顶指针,从而完成栈的切换。每个线程使用自己的栈存储入口事件标签,如图 5 所示。如此一来就保证了程序在正确执行时,标签匹配不会出错;而当发生控制流错误时,会出现匹配不一致的情况,从而检测到错误。

为了尽可能小地影响星载操作系统的实时性,可以对控制流错误检测方法进行优化。程序运行起来后,在操作系统内核中只有少数函数是反复运行的,其他函数可能只调用一次。确定这些反复运行的函数,加入检测语句,而对于不经常执行的函数则不再加入检测语句,达到减少软加固开销的目的。系统运行时经常调用的函数有:任务调度器、软中断调度器、硬中断管理程序、软件定时器处理函数、空闲任务循环、硬/软中断及任务的开关函数、队列管理函数等。在这些关键路径中加入检测语句,能保证较高的错误覆盖率,又避免了为每一个函数加入检测语句带来的开销。

3　控制流错误检测方法验证

为了测试控制流错误检测的效果,采用故障注入的方式验证软加固后程序的控制流错误检测能力。

实验硬件环境选取合众达国际的 SEED-DEC138 平台。该平台的主处理器为 TI 公司生产的 OMAP-L138,包含一个 TMS320C6748 DSP 核及一个 ARM 核,本文仅使用其 DSP 核。实验软件环境为 TI 公司的 Code Composer Studio V4.1.2(简称 CCS V4.1.2),运行在宿主机上。

在本实验中,应用程序可视为操作系统的 bechmark 程序。所选 benchmark 应是比较复杂的多任务程序并且使用操作系统提供的主要功能。选取 SYS/BIOS User's Guide 中的一个多任务示例程序[8],并对其进行扩充,基本覆盖了 SYS/BIOS 中的各项功能。

SYS/BIOS 完全基于一种称为实时软件构件(Real-Time Software Component, RTSC)的技术开发。RTSC 具有一套完整的开发工具链,使用 Expanded C 语言。Expanded C 由 C 语言和 XDCscript 语言组成,其中 XDCscript 是一种类似 Javascript 的脚本语言。RTSC 引入包、接口和模块的概念,并提出构件生产者和构件消费者两种角色。SYS/BIOS 使用 RTSC 开发,将各部分模块化。其功能裁剪和扩展更方便,作为构件供应端供消费者使用。

实验具体过程为:首先在宿主机文件系统里找到 ti.sysbios 包(SYS/BIOS 全部在此包中实现)所对应的文件,在其中添加修改相关代码,实现控制流错误检测算法;然后,用 XDCscript 语言编写 package.bld(类似于 Makefile)文件并使用 xdc 命令进行打包。

接下来,在 CCS V4.1.2 中新建工程,将修改后的 ti.sysbios 包导入,作为 RTSC 构件供应端。同时,在工程中导入已经编写好的 benchmark 程序。在 CCS 中执行 Build 命令,经配置、编译、连接后生成.out 可执行程序。

利用 CCS 连接 SEED-DEC138 平台,并将.out 程序加载到其上运行。在 CCS 中可以查看并修改 SEED-DEC138 平台中的寄存器值,通过对.out 文件在执行时随机写入一次二进制位翻转来实现故障注入。以修改 PC 寄存器为主,模拟注入控制流错误。

根据对程序的不同影响把注入的故障分为 5 大类:

1)结果错误。程序正常执行结束但结果错误。

2)运行超时。注入的故障导致程序执行超时,例如程序陷入死循环、系统崩溃等。

3)软加固算法检错。控制流错误被加入到操作系统中的控制流错误检测算法检测到。

4)操作系统检错。控制流错误被操作系统原有的内部检错机制检测到,如段错误等。

5)结果正确。注入的故障不影响程序的正常执行,执行结果依然正确。

对加入控制流错误检测算法前后的可执行程序分别进行 300 次故障注入实验,每次注入一个故障,得到的实验数据如表 1 所示。

表 1　实验结果

程序类型	各种故障类型所占故障注入总数的比例					控制流错误覆盖率/%
	结果错误率/%	运行超时率/%	软加固算法检错率/%	操作系统检错率/%	结果正确率/%	
未加固程序	15.3	18.3	0.0	43.4	23.0	66.4
软加固程序	4.0	4.7	38.3	34.3	18.7	91.3

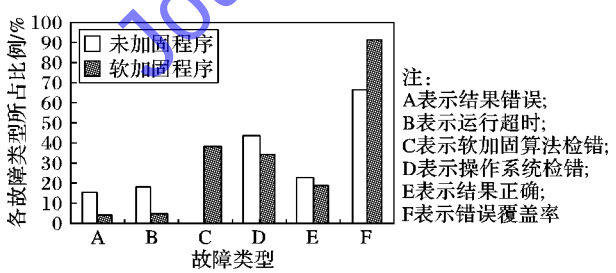

图 6　实验结果

错误覆盖率为 3)、4)、5)这 3 项所占比例之和,是指错误在发生后被检出或不影响程序运行结果的情况所占实验总次数的比率。这部分错误可以利用恢复机制及时处理。而"结果错误"和"运行超时"是未被检出的错误,直接导致错误的执行结果。从图 6 中可以看出:在加入控制流错误检测方法后,控制流错误覆盖率有了明显提高,从 66.4% 提高到 91.3%,同时结果错误和运行超时所占比例有了明显下降。这主要是因为及时检测到控制流错误,防止了其进一步传播,对程序造成危害,比如错误的跳转到空闲任务的 loop 中,使程序陷入死循环等。

4　结语

本文提出了一种控制流错误检测方法,对星载操作系统可能出现的控制流错误进行同步检测,通过标签存储栈记录并检查各线程的函数调用序列达到检错的目的。虽然相比应用程序所达到的 95% 乃至 98% 的控制流错误覆盖率,本方法还存在差距。但由于在较高层次上进行,而不必在汇编级密

Journal of Computer Applications 计算机应用

集地插入检查指令,本方法能更好地保证操作系统的实时性,而且错误覆盖率相比软加固前也有了明显提高。同时该方法可以通过改变所选加固函数的数目,达到更好地在性能与错误覆盖率之间做出权衡的目的。

操作系统的复杂性决定了需要多种软加固方法配合使用来提高错误覆盖率。在下一步工作中,拟研究一种异步周期检查的控制流错误检测方法,以进一步提高星载操作系统的可靠性。

参考文献:

[1] XU J, QIAO Q, XIONG M, *et al*. Software fault-tolerance techniques for transient faults [J]. Computer Engineering and Science, 2011, 33(11): 132 – 139. (徐建军, 谭庆平, 熊荫乔, 等. 面向瞬态故障的软件容错技术[J]. 计算机工程与科学, 2011, 33(11): 132 – 139.)

[2] OH N, SHIRVANI P P, McCLUSKEY E J. Control-flow checking by software signatures [J]. IEEE Transactions on Reliability, 2002, 51(1): 111 – 122.

[3] OH N, SHIRVANI P P, McCLUSKEY E J. Error detection by duplicated instructions in super-scalar processors [J]. IEEE Transactions on Reliability, 2002, 51(1): 63 – 75.

[4] NICOLESCU B, IGNAT N, SAVARIA Y, *et al*. Analysis of real-time systems sensitivity to transient faults using microC kernel [J]. IEEE Transactions on Nuclear Science, 2006, 53(4): 1902 – 1909.

[5] IGNAT N, NICOLESCU B, SAVARIA Y, *et al*. Soft-error classification and impact analysis on real-time operating systems [C]// DATE 2006: Proceedings of the 2006 Conference on Design, Automation and Test in Europe. Piscataway: IEEE, 2006: 182 – 187.

[6] NEISHABURI M H, DANESHTALAB M, KAKOEE M R, *et al*. Improving robustness of Real-Time Operating Systems (RTOS) services related to soft-errors [C]// AICCSA 2007: Proceedings of the 2007 IEEE/ACS International Conference on Computer Systems and Applications. Piscataway: IEEE, 2007: 528 – 534.

[7] YANG M, WANG H, ZHANG Y, *et al*. Design of radiation-hardened real-time operating system for pico-satellites [J]. Journal of Zhejiang University: Engineering Science Edition, 2011, 45(6): 1021 – 1026. (杨牧, 王昊, 张钰, 等. 抗辐射加固的皮卫星用实时操作系统设计[J]. 浙江大学学报: 工学版, 2011, 45(6): 1021 – 1026.)

[8] Texas Instruments. TI DSP/BIOS real-time operating system v6. x user's guide [DB/OL]. [2013-06-02]. http://downloads.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/sysbios/6_21_01_16/exports/docs/docs/Bios_User_Guide.pdf.

[9] HARI S K S, ADVE S V, NAEIMI H. Low-cost program-level detectors for reducing silent data corruptions [C]// DSN 2012: Proceedings of the 2012 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks. Piscataway: IEEE, 2012: 1 – 12.

[10] XU J, TAN Q, LI J, *et al*. An extendable control flow checking method based on formatted signatures [J]. Journal of Computer Research and Development, 2011, 54(4): 638 – 646. (徐建军, 谭庆平, 李建立, 等. 一种基于格式化标签的可扩展控制流检测方法[J]. 计算机研究与发展, 2011, 54(4): 638 – 646.)

[11] LI J, TAN Q, XU J. A software-implemented configurable control flow checking method [C]// PAAP 2010: Proceedings of the 2010 Third International Symposium on Parallel Architectures, Algorithms and Programming. Piscataway: IEEE, 2010: 199 – 205.

[12] LI J, TAN Q, XU J. Reconstructing control flow graph for control flow checking [C]// PIC 2010: Proceedings of the 2010 IEEE International Conference on Progress in Informatics and Computing. Piscataway: IEEE, 2010: 527 – 531.

(上接第 1407 页)

[3] CAI W, TAI Y, LIU Q, *et al*. Memory virtualization on MIPS architecture [J]. Journal of Computer Research and Development, 2013, 50(10): 2247 – 2252. (蔡万伟, 台运方, 刘奇, 等. 基于MIPS架构的内存虚拟化研究[J]. 计算机研究与发展, 2013, 50(10): 2247 – 2252.)

[4] BECKER M, BALDIN D, KUZNIK C, *et al*. XEMU: an efficient QEMU based binary mutation testing framework for embedded software [C]// EMSOFT '12: Proceedings of the Tenth ACM International Conference on Embedded Software. New York: ACM, 2012: 33 – 42.

[5] LIANG A, GUAN H, LI Z. A research on register mapping strategies of QEMU [C]// Proceedings of the 2nd International Symposium on Intelligence Computation and Applications. Berlin: Springer, 2007: 168 – 172.

[6] WEN Y, TANG D, QI F. Register mapping and register function cutting out implementation in binary translation [J]. Journal of Software, 2009, 20(S): 1 – 7. (文延华, 唐大国, 漆锋滨. 二进制翻译中的寄存器映射与剪裁的实现[J]. 软件学报, 2009, 20(S): 1 – 7.)

[7] CAI S, LIU Q, WANG J, *et al*. Optimization of binary translator based on GODSON CPU [J]. Computer Engineering, 2009, 35(7): 280 – 282. (蔡嵩松, 刘奇, 王剑, 等. 基于龙芯处理器的二进制翻译器优化[J]. 计算机工程, 2009, 35(7): 280 – 282.)

[8] LIAO Y. Dynamic binary translation modeling and parallelization research [D]. Hefei: University of Science and Technology of China, 2013. (廖银. 动态二进制翻译建模及其并行化研究[D]. 合肥: 中国科学技术大学, 2013.)

[9] DENG H. Reorganization and optimization of the backend code in dynamic binary translation [D]. Shanghai: Shanghai Jiao Tong University, 2011. (邓海鹏. 动态二进制翻译后端代码热路径的重组优化[D]. 上海: 上海交通大学, 2011.)

[10] SUN T. Branch analysis and optimization in dynamic binary translation [D]. Shanghai: Shanghai Jiao Tong University, 2010. (孙廷韬. 动态二进制翻译中跳转分析与优化[D]. 上海: 上海交通大学, 2010.)

[11] CALLEJA D. Linux 3.1 [EB/OL]. [2011-10-24]. http://kernelnewbies.org/Linux_3.1.

[12] QEMU. Change Log/1.2 [EB/OL]. [2012-09-05]. http://wiki.qemu.org/ChangeLog/1.2.

[13] RICHARD M S. Using the GNU compiler collection [EB/OL]. [2013-05-06]. http://gcc.gnu.org/onlinedocs/gcc-4.6.2/gcc.pdf.

Journal of 《计算机应用》 Computer Applications