嵌入式实时操作系统的分析评测方法①

张文君, 陈香兰, 李 曦

(中国科学技术大学 计算机科学与技术学院, 合肥 230001)

摘 要: 使用 WCET(Worst-case execution time)分析工具 Bound-T, 分析典型实时操作系统(RTMES 和 uClinux)的关键模块代码, 在系统运行在硬件上之前分析其机器码, 给出整体系统的最坏执行时间. 在系统的 WCET 达到要求之后, 再通过实验使用 benchmark, 评测操作系统的典型实时性能指标, 给出两个嵌入式实时操作系统的实时性能对比, 并分析 RTEMS(Real Time Executive for Multiprocessor Systems)的优势所在.

关键词: 嵌入式操作系统; 实时; 最坏执行时间

Analysis and Evaluation of Embedded Real-Time Operating System Methods

ZHANG Wen-Jun, CHEN Xiang-Lan, LI Xi

(School of Computer Science and Technology, University of Science and Technology of China, Hefei 230001, China)

Abstract: In this paper, we use WCET(Worst-case execution time) tool, Bound-T, to analyze the key modules of typical real-time operating system (RTEMS and uClinux) affecting real-time performance. The WCET tool analyzes machine code to give the whole system's WCET before it runs on hardware. When WCET reaches our requirements, we evaluate the system real-time performance through benchmark. We compare the performance of two real-time operating systems and explain the advantages of RTEMS (Real Time Executive for Multiprocessor Systems).

Key words: embedded operating system; real-time; worst-case execution time

1 引言

实时操作系统(Real-time Operating System)的发展和应用已经涉及到我们生活的方方面面, 所应用到的领域包括消费电子领域、通信领域、工业控制、汽车电子、医疗器械领域、国防、航空航天领域, 使得我们的生活更加的智能化和网络化[1].

实时操作系统不同于其他操作系统, 它们对时间都有着严格甚至近乎苛刻的要求, 如果不能保证实时性将会导致灾难性的后果. 实时性不仅要体现在系统要运行的快, 还体现在系统运行的时间具有确定性. 对操作系统实时性的保证不仅仅要在代码层次, 还需要通过分析评测得出具体的时间给出实时性的一个量化依据.

对最坏执行时间(WCET)的分析研究已经近三十多年的历史了, 最早开始研究最坏执行时间的研究者之一是操作系统之父 Alan Shaw[3], 目前有一大批学者和很多的研究机构已经在最坏执行时间分析上做了大量的工作, 但是由于它的复杂性及实时系统整体本身的复杂性, 在它的研究领域还存在着很多待解决的问题.

目前, 对于嵌入式实时操作系统的评测方法有很多种, 如 Whetston[2]、Dhrystone[5]等评测方法, 这些评测方法可以提供很精确的评测基准, 但都集中在研究如何评测运行系统的实时性.

从软件工程的角度来讲, 我们在设计实现操作系统的整个过程中都需要保证系统的实时性. 我们不仅需要对运行系统提供精确的评测基准, 我还需要在系统运行之前给出系统的 WCET 分析. 这样可以保证运行系统的安全性和提前发现系统的实时性能潜在问题, 也可以给实验评测提供量化的实时性能上限值.

本文先使用 WCET 分析工具 Bound-T, 分析对比 RTEMS 和 uClinux 各个关键模块代码的最坏执行时间.

① 收稿时间:2014-04-28;收到修改稿时间:2014-05-19

在操作系统源码中抽取出对实时性影响最大的模块代码, 编译链接产生目标代码, 使用 WCET 分析工具计算出最坏执行时间.

然后将 RTEMS 或者 uClinux 在硬件上运行起来, 使用 benchmark 测试对应的实时性能指标, 此时得到的测试结果应该要优于 WCET 分析工具得出的结果, 通过此实时性能指标的评测结果可以评判 RTEMS 和 uClinux 的实时性能优劣程度.

2 评测操作系统和实验评测方法的选择

2.1 分析评测操作系统的选择

RTEMS 是一个为嵌入式系统设计的自由的开源实时操作系统, 并且它是无版税开源的. RTEMS 最早用于美国国防系统, 现在由 OAR 公司负责版本的升级和维护, 无论在航空航天、军工, 还是民用领域 RTEMS 都有着极为广泛的应用.

uClinux 是嵌入式 Linux 的一个典型之作, 同时 uClinux 是一个完全符合 GNU/GPL 的操作系统, 完全开放代码, 现在由 Lineo 公司维护和支持. uClinux 和 Linux 有着相同的基础架构, 且可移植性很强, 可方便的移植到多种处理器平台.

表 1 RTEMS 和 uClinux 基本特点对比

操作系统	uClinux	RTEMS
内核代码	开源, 版本多	开源
内核最初设计来源	从 Linux 剪裁改造而来	为美国军方所定制
软件成本	免费	免费
内核内存保护	无	无
内核尺寸	100KB+	30KB
多任务模型	多进程多线程, 进程可以有不同优先级	单进程多线程
内存管理	不支持虚拟内存	不支持虚拟内存
系统 API	丰富	丰富

选择 RTEMS 和 uClinux 作为本文分析评测的两个嵌入式操作系统, 最重要的原因是它们都是开源的, 可以方便的获取源代码, 并且它们是典型的实时操作系统.

2.2 实时性评测方法的选择

评测一个实时操作系统的实时性的评测方法有很多种:

(1) Whetston 评测方法

Wichman[2]在 1972 年提出 Whetstone 评测方法, 基于运算和指令, 主要关注浮点指令性能. Whetstone 评测使用数组参数传递、浮点数运算、条件转移、子程序调用、数组元素访问、数组引用等模块来评估系统的计算能力. 给各个模块赋予一定的权值, 计算完成后分别加权得到最终评估结果.

Whetstone 使用硬件无关的 WIPS(Whetstone Instructions Per Second)为单位来度量测试结果. 系统调用模块评测用例, 每完成一次主循环, 记录一个 Whetstone 指令, 单位时间内完成 Whetstone 指令的数量为单个模块的测试结果.

(2) Dhrystone 评测方法

R. EWeicker 等于 1984 年提出 Dhrystonetl[5]评测方法, 重点关注整数的计算性能, 评测代码所占空间比较小. Dhrystone 评测方法选择高级语言中频繁使用到的赋值、函数调用、分支控制等典型操作来评测一个系统的性能. Dhrystone 评测方法不考虑超标量 RISC 设计、VLIW(Very Long Instruction Word)、TCP/IP、图形用户界面、多媒体应用. Dhrystone 所使用的评测工具是开源的, 虽然出现的比较早, 但是 Dhrystone 评测工具一直以来被广泛采用.

与 Whetstone 一样, Dhrystone 评测方法不使用常见的 MIPS(Million Instructions Per Second)作为性能的度量指标, 因为对于不同的 CPU 架构, 如 RISC、CISC, 完成同一个任务所需要的 MIPS 数不同. Dhrystone 评测方法使用的性能度量单位为 DIPS(Dhrystone Instructions Per Second), 每运行一次主程序循环记录一个 Dhrystone 指令, 用单位时间内系统所完成的任务数量, 来表征系统的性能.

(3) Hartstone 评测方法

Nelson 在 1989 年提出 Hartstone[6]评测方法, 用于综合评估硬实时系统. Hartstone 评测方法拥有五个的测试任务: 频率不变同步处理不定期周期性任务、频率不变同步处理周期性任务、频率不变不定期周期性任务、频率变化周期性任务、频率不变周期性任务. 每一个测试任务的测试结果有且仅有两种情况: 满足和不满足时间约束. 每一个测试任务包含一个子任务集, 用于定义负载、期限、周期. 从评测子任务集开始, 满足任务时间约束后, 更改任务集参数进一步评测, 超过响应时间或者没有满足任务期限则测试结束. 在最大负载情况下并且任务参数符合要求时得出测试结果.

(4) Rhealstone 评测方法

Rhealstone[9]由 Kar 等人, 于 1989 年提出, 是针对实时操作系统的综合评估方法. 该评测方法提出了任务抢占时间、任务切换时间、中断延迟时间、死锁解除时间、信号量混洗时间和消息延迟时间六项指标, 涵盖了任务、同步、中断三个实时性主要方面. Rhealstone 评测方法实现简单, 可移植性强, 被广泛使用, 并随着实时操作系统的发展不断地改进和发展.

Rhealstone 指标一般以 us、ms 或者 s 为单位, 最终结果是通过多次测量得出的. 如果 t1、t2、t3、t4、t5、t6 分别是以上六个指标的测量结果, 计算出它们的平均值 t0, 对 t0 还可以根据需要进一步加权分析, 得到更全面的评测结果.

(5) 评测方法的选择

从以上 4 种常见的评测方法, Whetston、Dhrystone、Hartstone 和 Rhealstone 评测方法中, 可以看到 Whetston 和 Dhrystone 评测方法主要是偏向硬件系统的综合评测, 主要关注计算的性能和指令的性能, Hartstone 则是专门用于硬实时系统的评测方法, Rhealstone 评测方法能够全面的给出实时性的评测结果, 特别是操作系统级的实时性能. 因此本文选择 Rhealstone 评测方法作为实验评测的方法.

3 操作系统实时性分析

3.1 WCET 分析

WCET, 最坏执行时间, 是指系统中的一个任务在特定硬件平台上执行所花费时间的最大值, 最坏执行时间是在硬实时系统调度分析中最重要的环节.

WCET 分析的方法主要分为静态分析和动态测量技术.

静态分析技术, 无需软件直接运行在硬件平台上, 通过分析程序可能的行为, 计算出软件的最坏执行时间.

动态测量技术, 是在实际硬件平台上, 使用输入集合, 测量出这个任务或者此代码模块的执行时间.

动态测量技术的输入集合直接决定最终的结果如何, 而且测量只给出整个程序的平均执行时间和最长执行时间, 我们并不清楚程序中各个代码模块对系统整体执行时间的影响. 动态策略方法试图对所有输入组合测量出执行时间, 但是我们不可能将程序的输入组合全部穷举出来, 随着系统功能的逐渐庞大, 程序的输入组合的大小是呈 O(n!)增长的, 显然对于硬实时系统, 动态测量技术的可靠性和可信度是远远不够的. 另外, 静态分析方法, 使用估值和控制路径方法, 结合处理器行为模型, 分析源程序和目标代码, 给出执行时间的上限值, 静态分析方法可以覆盖代码的所有可能执行路径, 并且可以针对 cache、流水线和分支预测建立分析模型. 再者, 静态分析技术可以在复杂体系结构上得到很好的发挥. 因此, 在本文中采用静态分析技术来分析 WCET.

3.2 WCET 分析工具

Bound-T 是 Tidorum Ltd 的一款用于静态分析的 WCET 分析工具, 使用 Bound-T 可以得到代码的执行时间、栈的使用率、控制流程图和调用关系.

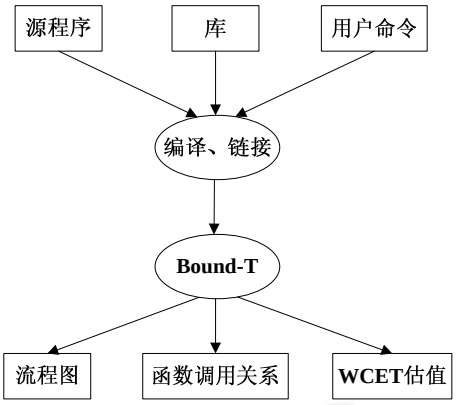

图 1　Bound-T 的工作原理图

2.3.1 Bound-T 的特点:

(1) Bound-T 是一个静态分析工具, 静态分析的一个特点是不需要代码运行在硬件平台上, 不需要搭建一个真实的测试环境.

(2) Bound-T 采用数据流分析的常规方法和数学模型, 查找循环计数器变量并设定它的初始值和最终值, 由此来确定循环重复的次数.

(3) 使用 Bound-T 分析的是可执行代码, 所以它能够独立于源代码的开发语言, 甚至可以处理 C 语言和汇编语言混合程序.

(4) 通过编写计算变量值、循环界限和调用执行次数的脚本, 这些脚本可以帮助用户进行复杂程序的分析, 但是不能全部自动分析, 有些还需要手动分析. 脚本可以使用文本形式编写, 修改脚本 Bound-T 将分析不同的执行方案, 比如分析程序对不同输入加载的响应时间.

(5) Bound-T 可以使用实时循环和调用架构创建程序的框架, 而不是通过虚拟循环估算结果. 还可以计算出整个 WCET 边界, 以确定符合这些预算的截止时间. Bound-T 最常使用的用途是在编码和测试阶段, 用户可以使用 Bound-T 验证每个代码模块的每次操作是否符合其时间预算, 程序不一定是已经编码完成, 也不一定是可执行的, 只需要编译链接通过即可. 最终, 用户可以在此基础上确定系统是否符合时间性能要求.

(6) Bound-T 可以模仿流水线时序, 而非采取模仿缓存的行为或其他动态功能的方法, 后者会因为时间不准确而导致失真.

3.3 WCET 分析结果

分别对 RTEMS 和 uClinux 的关键任务调度部分, 使用 Bound-T 测试.

选取的操作系统为: RTEMS 4.10.2 和 uClinux-20121024.

以 RTEMS 为例, 选取任务模块函数: 创建任务(rtems_task_create)、删除任务(rtems_task_delete)、挂起任务(rtems_task_suspend)、唤醒任务(rtems_task_resume); 选取信号量模块函数: 获取信号量(rtems_semaphore_obtain)、释放信号量(rtems_semaphore_release); 选取消息模块函数: 发送消息(rtems_message_send)、接受消息(rtems_message_receive).

3.3.1 影响 Bound-T 分析结果的因素(以 RTEMS 为例)

(1) 运行队列管理

RTEMS 使用基于优先级、抢占算法为任务分配处理器, 对于优先级相同的任务采用时间片轮转的方法调度. 此调度策略的目的, 为了保证运行在处理器上的任务优先级始终是就绪态任务优先级的最大值. 在此调度代码模块中有两个循环体, 一个负责按照任务优先级把任务 ID 放入运行队列上, 另一个负责在运行队列上查找给定优先级的任务. 这两个循环体的循环上限是 MAX_READY_TASK, 最坏的情况下 MAX_READY_TASK 等于 MAX_TASK(MAX_TASK 是系统任务的最大值). 静态分析涉及到运行队列, 对于此模块的时间分析得出的最坏执行时间很有可能被高估.

(2) 堆管理

RTEMS 在任务创建和任务删除等操作中, 使用的是动态内存分配策略, 使用首次匹配最小合适内存块的方法. 内存堆被划分为已用块和未用块, 堆管理策略使用双向链表管理未用块和已用块. 在需要未用块的时候, 内存分配函数按照最小匹配算法, 在双向链表上搜索适合的内存块. 最坏的情况是, 双向链表上存放的内存块是未用块和已用块连续间隔的(如下图), 并且他们的大小都是 MIN_SIZE, 所以最坏情况下要搜索 HEAP_SIZE/(2*MIN_SIZE)次, HEAP_SIZE 是系统分配给动态内存堆的最大内存. 所以, 动态分配策略的不确定性会直接影响到最坏执行时间的分析.

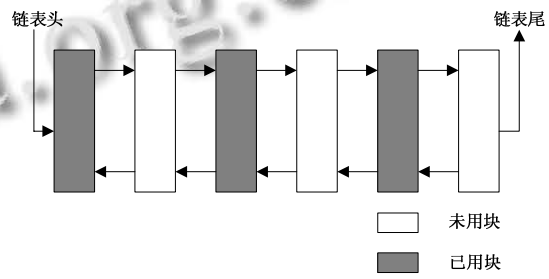

图 2 双向链表

(3) 动态函数调用

所谓动态函数调用是指, 调用函数的时候不是直接调用, 而是通过函数指针间接调用. RTEMS 允许用户添加扩展例程, 比如任务的创建, 新的扩展例程是被动态的添加的. 处理这些扩展例程, 有两种方法, 第一种是禁止使用用户扩展, 这种方法极大的限制了系统的功能; 第二种方法是标识出每个动态函数. 在进一步深入的研究代码之后发现, RTEMS 所涉及到的用户扩展例程, 包括任务创建、任务删除、任务开启和任务重启等, 这些例程虽然是动态系统调用, 但是在系统运行之前就可以确定好了, 因此系统已经帮我们标识出这些动态函数了. 根据之前所述, 本文测 WCET 的时候, 在需要的地方使用函数名替换函数指针去显式的标识动态函数.

(4) 中断

任务在运行的时候, 一个中断随时都会发生, 中断会将正在运行的任务打断, 由于本文讨论的是静态 WCET 分析, 在静态 WCET 分析中, 中断对任务运行的影响仅在于, 调度器的主循环什么时候被打断. 在 RTEMS 中调度器的关键函数是_Thread_Dispatch(), 在本文评测方法中只有时钟中断, 时钟中断的时间间隔是已知的. 所以在本文中可以控制中断对静态

WCET 分析的影响.

(5)流水线

在流水线处理器上, 指令执行时间依赖其它指令在流水线上的执行时间. 为了更好的分析流水线处理器, Bound-T 对流水线状态进行建模, Bound-T 使用控制流图来表征流水线的状态, 一个流水线的子状态表示一个指令集合, 当流水线发生停顿的时候, 更容易在控制流图上定位出哪个位置发生了停顿.

3.3.2 分析结果

以任务、信号量和消息模块的分析评测举例, 使用-assert 选项的 Bound-T 命令能得到: 各个函数内部包含的循环次数, 每个函数被调用所用时间和 main 函数所花费的总时间. Bound-T 使用机器周期数作为衡量运行时间的基本单位.

使用加-table 选项的 Bound-T 命令得到输出结果经整理得出如下表格, 其中 min 和 max 的含义是指, 一个函数每次被调用所用时间的最小和最大时间. 表 2、表 3 和表 4 列出 Bound-T 所测 RTEMS 中任务、信号量和消息的分析结果, 其中运行时间的单位是机器周期数, 根据机器周期长度计算汇总得出表 5 的结果.

表 2　RTEMS 的 Bound-T 的任务分析结果

subprogram	min	max	calls	runtime
main	6412	6412	1	6412
rtems_task_create	510	557	20	525
rtems_task_delete	4523	4788	20	4613
rtems_task_suspend	369	430	20	408
rtems_task_resume	545	637	20	590

表 3　RTEMS 的 Bound-T 的信号量分析结果

subprogram	min	max	calls	runtime
main	2619	2619	1	2916
rtems_semaphore_obtain	874	1197	20	1005
rtems_semaphore_release	865	1096	20	908

表 4　RTEMS 的 Bound-T 的消息分析结果

subprogram	min	max	calls	runtime
main	7077	7077	1	7077
rtems_message_send	5501	5813	20	5619
rtems_message_receive	1088	1329	20	1212

表 5　RTEMS 和 uClinux 分析结果汇总

模块函数	RTEMS(us)	uClinux(us)
创建任务	52.5	1430
删除任务	461.3	1690
挂起任务	40.8	1880
唤醒任务	59.0	1300
申请信号量	100.5	1260
释放信号量	90.8	1230
发送消息	561.9	3410
接受消息	121.2	850

3.3.3 栈使用的分析

使用-stack 和-stack_path 两个参数命令能得到所分析模块的函数调用关系, 能知道函数调用栈的最大深度. 以申请信号量为例.

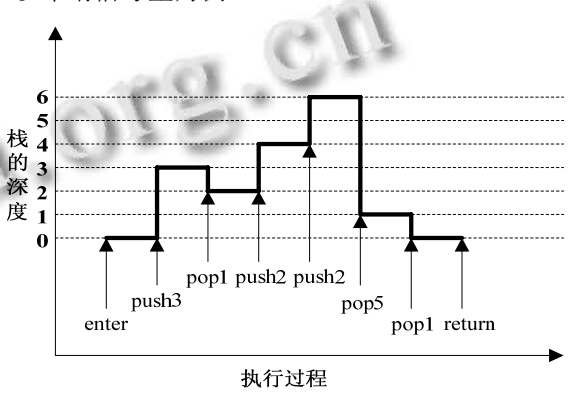

图 3　函数调用的栈分析图

4　实时性能的评测

本文对运行在 ARM7(100Mhz)上的 RTEMS 和 uClinux, 利用定时器中的数值, 在计时开始的时候和计时结束的时候分别读取定时器对应寄存器的值, 计算出时间差. 按照 Rhealstone 评测方法, 针对任务切换时间、任务抢占时间、中断延迟时间、信号量混洗时间、死锁解除时间、消息延迟时间和八个系统调用函数, 分别评测得出结果.

表 6　实时指标的时间

实时指标	RTEMS(us)	uClinux(us)
任务切换时间	101.2	300
任务抢占时间	175.1	2900
中断延迟时间	35.5	450
信号量混洗时间	630.0	5100
死锁解除时间	212.3	610
消息延迟时间	183.7	750

表 7　系统调用的时间

模块函数	RTEMS (us)	uClinux(us)
创建任务	36.4	1210
删除任务	328.3	1500
挂起任务	25.0	1250
唤醒任务	43.3	850

申请信号量	80.2	600
释放信号量	78.6	830
发送消息	464.3	2900
接受消息	90.7	340

5 Bound-T分析和评测结果对比

为了更加直观的看出 Bound-T 分析和 Benchmark 实验的结果, 下面将表格数据转换成图的形式.

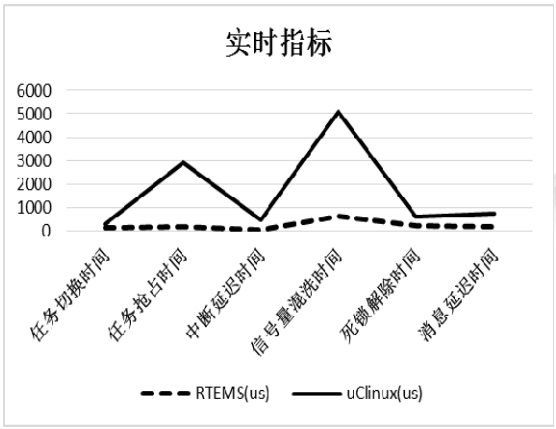

图 4 实时指标时间对比图

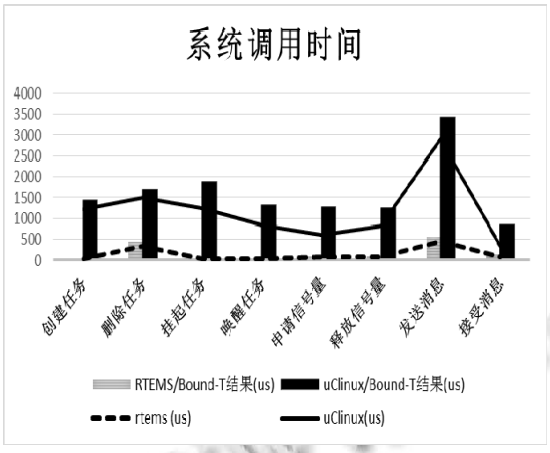

图 5 系统调用函数时间对比图

下面对所得结果进行分析:

(1)考虑到影响 WCET 分析的因素, Bound-T 所测的系统调用函数时间, 要比 Benchmark 所测时间偏大. 堆管理使用到的双向队列、时钟中断和流水线是影响到 Bound-T 分析的三个主要因素, Bound-T 分析时候, 不仅记录每个函数执行的机器周期数, 在得出最后结果的之前会累加这些影响因素, 如果 Bound-T 无法计算出影响因素的准确值, 就会按照最坏情况累加, 所以 Bound-T 所测时间要偏大.

(2)Bound-T 栈分析, 从图 3 中体现出所分析函数运行状态和运行过程, 更重要的是可以明显的看出一个模块函数中栈最大深度. 特别对于内存紧缺的实时系统, 栈分析是必不可少的一个工作.

(3)相比较 uClinux, RTEMS 的六个实时性能指标表现都很好, 基本都是微秒级别的. 系统调用函数的时间表现, RTEMS 的性能也很出色.

(4)在图 4 和表 5 系统调用时间中, 按照功能比较, RTEMS 任务模块和信号量模块的时间基本没有较大的差别, 常用的创建任务和挂起任务函数的时间都比申请信号量和释放信号量的时间要低, 而 uClinux 的任务模块时间比申请信号量和释放信号量的时间都要高. 在操作系统运行过程中任务作为系统运作的最小单元, 需要对它进行频繁的操作, 任务模块的时间花费太多将对系统的实时性影响很大. 从实时性的六个指标比较中能看出来, RTEMS 的任务切换时间和任务抢占时间整体要优于 RTEMS 其他的时间指标. 任务模块和消息模块也有类似的体现.

(5)从图 4 可以看出, RTEMS 的各个实时指标时间线走向趋于平滑, 系统的整体更加稳定, 没有过大的波动, 所以在系统稳定性上, RTEMS 也占有优势.

6 总结

本文对于实时操作系统的开发过程, 按照软件工程的思想, 给出一个保证实时性的方案. 通过 Bound-T 从目标代码中分析 WCET, 然后在硬件平台上, 用 Benchmark 测试各个实时指标. 如果 WCET 分析出的时间已经远远超出预期时间, 此时需重新审查代码是否有缺陷, 如果用 Benchmark 所测试的时间比 WCET 分析出的时间高, 很可能模块交互部分出现问题, 此时能容易定位问题所在位置. 本文所提方案可以提前发现开发过程中的问题. 在实际运行之前给出实时性的一个时间上限值, 从工程的角度保证实时系统的开发. 在 Benchmark 测试部分, 可以进一步增加任务数, 得出更加全面和更加精确的结果. 虽然 uClinux 可配置性非常高, 应用非常广泛, 但是在实时性要求严格的环境下, RTEMS 要优于 uClinux.

参考文献

1 Liu J. Real-Time Systems. Beijing: Higher Education Press, 2000.

2 Wichman BA. Whetstone: A synthetic benchmark. Computer Journal, 1976, 19(1): 43–49.

3 Heckmann R, Langenbach M, Thesing S, Wilhelm R. The influence of processor architecture on the design and the results of WCET tools. Proc. of the IEEE Special Issue on Real-Time Systems, July 2003, 91(7): 1038–1054.

4 Barabanov M. A Linux-based Real-Time Operating System[Thesis]. Socorro: New Mexico Institute of Mining and Technology, 1997.

5 Weicker R. Dhrystone: A synthetic systems programming benchmark. Communication of the ACM(CACM), 1984, 27(10): 1013–1030.

6 Weiderman NW. Hartstone: Synthetic Benchmark Requirements for Hard Real-Time Applications[Master Thesis]. Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 1989.

7 Sha L. Real-time in the real world. New York, USA: ACM Workshop on Strategic Directions in Computing Research, 1996.

8 Morelli E, Smith M. Real-time dynamic modeling: data information requirements and flight-test results. Journal of Aircraft, 2009, 46(6): 1894–1905.

9 Kar R, Porter K. Rhealstone: A real-time benchmarking proposal. Dr. Dobbs Journal, 1989, 14(2): 14–24.

10 Krishna C, Shin K. Real-Time Systems. New York: The McGraw-Hill Companies, Inc., 1997.

11 Stodolsky D, Chen BJ, Bershad BN. Fast Interrupt Priority Management in Operating System Kernels[Technical Report], CMU-CS-93-152, School of Computer Science Carnegie Mellon Unversity. 1993.