RTX 实时效果测试及应用①

The test of RTX real – time effection and application

田 昊　潘 清　（装备指挥技术学院 信息装备系 北京 101416）

摘要：本文主要针对 RTX（Real – Time eXtension）嵌入式实时平台的实时优化效果进行测试验证。通过在 Windows 系统中安装 RTX 平台，我们对一些关键的实时性能进行了测试对比。测试范围主要包括线程的切换时间，异常处理等待时间，高优先级线程抢先时间以及信号量交替延迟时间方面，并对 RTX 进程的高优先级特性进行了验证。另外，还对 RTX 和图形用户界面结合的使用方法进行了讨论。

关键词：RTX　实时　优化

1　引言

当今，嵌入式实时应用有着宽广的应用领域。例如基于 GUI 的用户操作界面，某些通信信息系统，以及有着严格时间要求的监控设备等等。对于这些相关应用的开发者和最终用户来讲，由于 Windows 2000 和 Windows XP 操作系统对硬件的广泛支持和拥有多种配套的应用软件，为开发和应用带来了巨大的方便，从而成为了可选的操作系统平台。

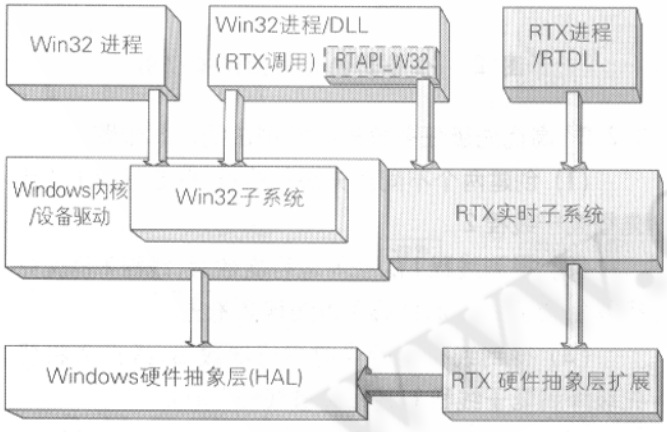

图 1　RTX 体系结构图

但是，它们的功能并不是针对实时性应用开发的，从而它们在实时领域的使用受到很大限制。采用嵌入式实时平台软件来提高系统的实时处理能力能较好地弥补它们的这一缺陷，从而较好地满足实时应用的相关要求。

2　RTX 实时系统分析

RTX 实时系统作为 Windows 操作系统内核体系的延拓，修改并扩展了整个硬件抽象层 HAL（Hardware Abstraction Layer），实现了独立的内核驱动模式，形成了与 Windows 操作系统并列的实时子系统 RTSS（Real – Time Sub System）。其体系结构如图 1。

RTSS 从概念上来讲类似于其他的 Windows 子系统（例如 Win 32，POSIX，WOW，和 DOS），因为它支持它自己的执行环境和应用程序接口 API（Application Program Interface）。但是 RTSS 的一大不同之处，就是它不使用 Windows 的调度策略，而是自己对实时线程进行严格调度管理。另外，在一个单处理器环境下，所有的 RTSS 线程调度发生在所有的 Windows 调度之前。

RTSS 也支持既能由 RTSS 进程，也能由 Win32 进程操作的进程间通信 IPC（InterprocessCommunication）对象，这就使得实时与非实时程序间的同步更容易。另外通过扩展的 HAL，RTX 有着自己中断管理机制，而且能够直接访问 I/O 硬件端口。

RTX 的 API 以 Win32 的 API 作为基础，因此能在 Win32 的经验、代码和开发工具的基础上，加快硬实时应用的开发效率。Win32 和 RTSS 进程同时支持完整的 RTX API，但二者却有着不同的响应时间和性能特

① 基金项目：部委级基金资助项目

征。

3 实时性能测试

3.1 测试指标

测量系统的实时性，有一些专门的测试方法，其中 Rehealstone 方法定义了以下 6 项指标来衡量实时系统的性能。

（1）任务切换时间。即系统在两个独立的、处在激活态并具有相同优先级的任务之间切换所需要的时间。

（2）高优先级任务抢先时间。即系统将控制从低优先级的任务转移到高优先级的任务所花费的平均时间。

（3）异常处理等待时间。即从 CPU 收到异常请求到执行异常服务程序所用的时间。

（4）信号量交替延迟时间。从一个任务释放信号量到另一个等待信号量的任务被激活所需要的时间。

（5）死锁解脱时间。指系统解开死锁所需要的平均时间。

（6）数据报通过率。一个任务调用实时系统的原语，把数据传送到另一个任务去时，每秒传送的字节数。

在上面提到的六项测试指标中，我们对前四项进行了对比测试。另外，还对 RTSS 线程的高优先级的特性进行了测试。

3.2 测试方法

图 2 所示是任务切换时间测试的流程图，测试前先测定线程本身开销时间。其他几项测试与它类似。

3.2.1 任务切换时间测试的基本步骤

（1）创建两个相同优先级的线程 1 和 2

（2）运行线程 1 得到线程 1 的当前运行次数后，立即切换到线程 2

（3）运行线程 2 得到线程 2 的当前运行次数后，立即切换到线程 1

（4）重复第 2 步和第 3 步，直到其中一个线程运行次数等于 1000000 停止

（5）得到 2 至 4 步所花费的全部时间和两线程分别的运行次数

（6）计算线程间的切换时间

这里设所花费的时间为 emplasedtime，，线程本身开销为 usetime，分别运行的次数为 runcount1 和 runcount2，则切换时间为（emplasedtime − usetime ∗（runcount1 + runcount2））/（runcount1 + runcount2 − 1）。以下几项与此类似。

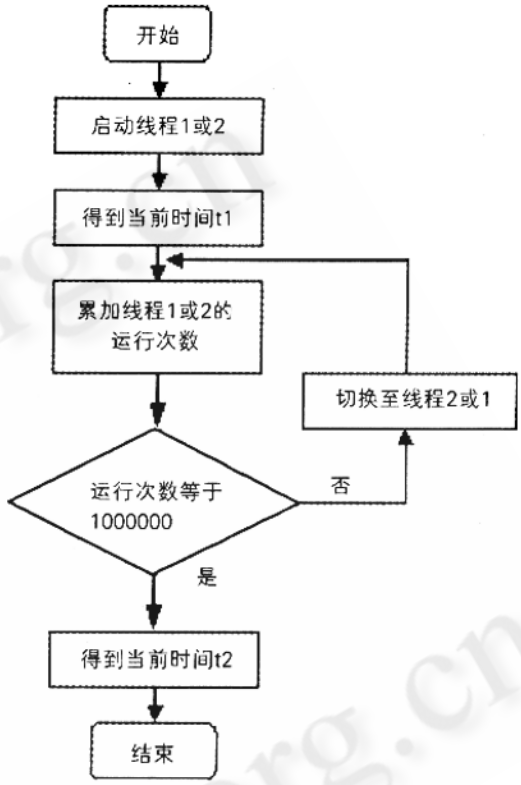

图 2　任务切换时间测试流程图

3.2.2 高优先级任务抢先时间测试的基本步骤

（1）创建两个不同优先级的线程，假定线程 1 的优先级高于线程 2

（2）得到线程 1 的当前运行次数后，线程 1 将优先级降到低于线程 2，线程 1 被线程 2 抢先

（3）得到线程 2 的当前运行次数后，线程 2 将优先级降到低于线程 1，线程 2 被线程 1 抢先

（4）重复第 2 步和第 3 步，直到其中一个线程运行次数等于 1000000 停止

（5）得到 2 至 4 步所花费的全部时间和两线程分别运行的次数

（6）计算高优先级线程抢先时间

3.2.3 信号量交替延迟时间测试的基本步骤

（1）创建信号量 1 和信号量 2

（2）建有相同优先级的线程 1 和 2，并分别请求获

得信号量 1 和信号量 2

(3) 释放一个信号量 1

(4) 线程 1 获得信号量 1,得到当前运行次数后,立即释放信号量 2

(5) 线程 2 获得信号量 2,得到当前运行次数后,立即释放信号量 1

(6) 重复第 4 和第 5 步,直到其中一个线程运行次数等于 1000000 停止

(7) 得到 4 至 6 步所花费的全部时间和两线程分别运行的次数

(8) 计算线程间信号量交替延迟时间

3.2.4 异常处理等待时间测试的基本步骤

(1) 创建运行一个线程

(2) 在线程中给出一个异常

(3) 异常处理中得出当前异常次数

(4) 重复第 2 和第 3 步,直到异常次数等于 1000000 停止

(5) 得到 2 至 4 步所花费的全部时间

(6) 计算异常处理等待的时间

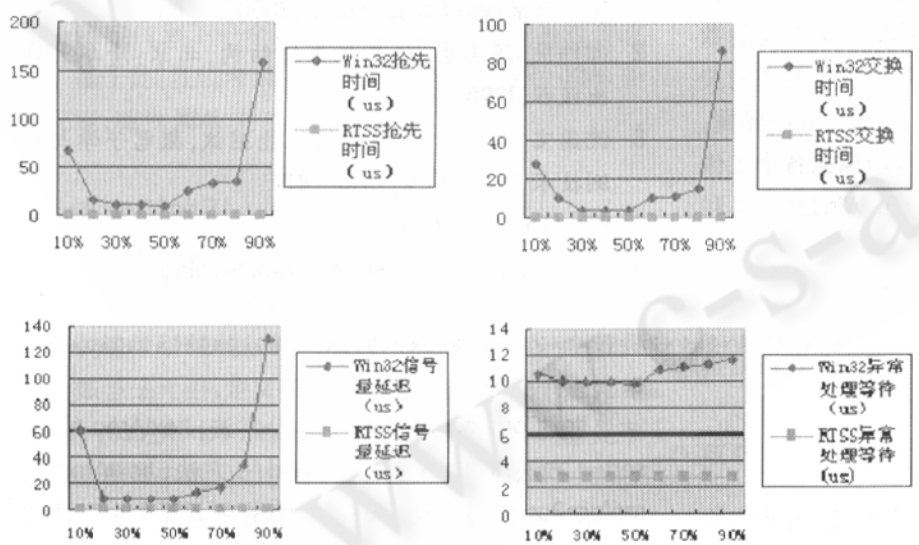

图 3　优先级特性测试效果对比图

3.2.5 高优先级的测试

为了测试 RTSS 线程的高优先级的特性,我们采用以下测试方法。

(1) 启动一个普通的 Win32 工作线程,作为背景任务

(2) 启动以上某一个测试指标的 Win32 测试程序,并得出相应的时间值

(3) 启动同一个测试指标的 RTSS 测试程序,并得出相应的时间值

(4) 重复(2)(3)两步数次,比较它们的测试结果

(5) 再对另一测试指标进行(2)(3)(4)步

(6) 如此重复进行直至所有指标都测试完毕

(7) 改变作为背景任务的程序中的某些参数的设定,使其运行时拥有不同的 CPU 占用率

(8) 这样再从第(1)步至第(7)步重复进行测试。(测试中 CPU 的占用率取为 10% 的整倍数)

3.3 测试结果

在硬件配置为 Intel 奔腾 2 处理器,内存为 128MB,操作系统为 Windows2000 的机器上分别运行调用 Win32API 函数的 Win32 程序和调用 RTX API 函数的 RTSS 程序,对于前四个指标,每个指标测试 10 次,得到 10 个时间值,然后取平均。结果如表 1。

对高优先级特性的测试结果(背景任务 CPU 占用率 10% – –90%)如以下曲线图 3 所示。

另外当 Win32 背景任务 CPU 占用率为 100% 时,Win32 进程和 RTSS 进程实时性能指标测试结果如表 2 所示。

3.4 测试结论

首先从表 1 的实时性能测试中我们可以看出,四个实时性能指标 RTSS 线程都明显优于 Win32 线程。也就是说,在 Windows 环境下,通过启动 RTX 进程,利用 RTSS 工作线程来完成我们的相应工作,能获得较好的实时性能,满足我们的实时应用的要求。

另外从图 3 和表 2 中关于 RTSS 线程高优先级的验证测试结果中,我们可以看出,随着 Win32 背景任务对 CPU 占用率的改变,Win32 线程的实时性能指标会受到较大影响,而 RTSS 线程的各项实时性能却几乎不受 Win32 背景任务的影响。也就是说,RTSS 线程的调用始终发生在 Win32 线程调用之前,即 RTSS 线程始终拥有较 Win32 线程更高的优先级。

表 1　实时性能测试对比

测试指标	切换时间(us)	高优先级抢先时间(us)	信号量交替延迟时间(us)	异常处理等待时间(us)
Win32 线程	1.72	4.23	4.15	9.10
RTSS 线程	0.17	0.75	1.36	2.80

表 2　CPU 占用率为 100% 时，优先级特性测试对比

测试指标	切换时间(us)	高优先级抢先时间(us)	信号量交替延迟时间(us)	异常处理等待时间(us)
Win32 线程	4700.00	5000.00	516.70	11.80
RTSS 线程	0.17	0.75	1.36	2.80

4　与 MFC 图形用户界面结合使用的讨论

虽然 RTX 进程拥有较好的实时性能，但其本身并不具备开发图形用户界面的条件。这对开发实时应用是一个很大的限制。但利用进程间的通行机制能较好地解决这一问题。

一般来讲图形用户界面的作用主要体现在两个方面：一是直接的、方便用户控制的功能；二是直观的数据显示的功能。若要开发拥有图形用户界面要求的实时应用，可先利用 MFC 相关机制开发图形用户界面，然后将具体的事务处理放到 RTX 进程所属的各个工作线程的代码中，二者的结合即可利用 Win32 进程和 RTX 进程间的通信来实现。

具体做法是，可利用图形用户界面的控制作用创建 Rt 信号量，在 RTX 进程中利用这些相应的信号量来控制所属工作线程的运行。即可达到图形用户界面对后台运行任务的控制作用。另外，可通过创建 Rt 共享存储区，利用共享存储区进行进程间数据的交换。如放入 RTSS 工作线程运行后得到的相关数据，在 Win32 进程中从此共享存储区中将这些数据取出，并相应地显示出来，即可实现直观的数据显示的功能。

5　结束语

通过对相关实时性能的测试，能够看出通过在 Windows 系统中嵌入 RTX 实时平台，实现独立的 RTSS，能较显著地提高系统的实时处理能力，较好地满足实时应用开发的需要。从而为诸如 Windows 这样的非实时系统应用于实时领域提供了可能。另外通过对与图形用户界面的结合使用的讨论和实现，进一步为其在实际中的使用打下了基础。

参考文献

1　Jeffery Richter[著]，王建华[译]。Windows 核心编程。北京：机械工业出版社，2000.

2　郑纬民、汤志忠，计算机系统结构，北京：清华大学出版社，1998.

3　沈国宝、刘松强，实时系统性能测试，微电子学与探测技术，2002，5[9]：416—419.

4　RTX SDK 帮助文档.

5　RTX 培训课程 。www. vci. com/training.