

## Windows 时钟机制的实时扩展研究

蒋善锋, 王丽芳, 蒋泽军

(西北工业大学 计算机学院, 陕西 西安 710129)

**摘 要:** 探讨了 Windows 中断机制, 设计了一个优先级高于系统时钟中断的实时时钟. 通过对实验测试数据的分析, 在没有其他中断的情况下, 新的时钟系统可以提供一个最小精度为  $100\ \mu\text{s}$  的实时时钟. 最后, 提出了一个利用新的实时时钟实现 Windows 实时拓展的设计实例.

**关键词:** Windows; 实时时钟; 中断

**中图分类号:** TP391

**文献标识码:** A

**文章编号:** 1000-7180(2013)08-0116-04

## Research of the Real-time Extension on Windows Clock Mechanism

JIANG Shan-feng, WANG Li-fang, JIANG Ze-jun

(School of Computer Science, Northwestern Polytechnical University, Xi'an 710129, China)

**Abstract:** The interrupt mechanism of Windows was discussed in this paper, and a real-time clock with higher priority than the system clock interrupt was designed by using the interrupt priority mechanism in Windows. Tests and analysis results show that the new clock system is able to provide a hard real-time clock with minimum accuracy of  $100\ \mu\text{s}$  under the condition that no other interrupts happen. Finally, a design based on the new clock system to improve the real-time performance of Windows is put forward.

**Key words:** Windows; real-time clock; interrupt

### 1 引言

Windows 操作系统具有优秀的图形界面和大量成熟的开发工具被广泛应用于很多领域. 由于 Windows 操作系统的一些机制存在实时缺陷, 不能用于对实时性有很高要求的工业应用. 文献[1-2]中提出了基于 APIC 的 Windows 实时改进方案, 这种方案利用硬件中断转去执行指定程序保证实时性, 但是没有给出完整的中断服务程序. 本文对 Windows 系统的硬件中断服务过程进行深入探索, 通过真实实验的方法验证了参考文献[1-2]所共同提出方案的可行性, 并分析得出了方案可行时的条件. 与此同时, 改进和完善了这种方案, 并给出了一个新的中断服务程序的设计结构.

本文所述的实时时钟方案可概述为: 通过启用 APIC 上的本地定时器, 并为其编写中断服务程序, 实现了一个中断优先级高于系统时钟中断的实时时

钟. 本文所论述的方法已经在 Microsoft Windows XP Service Pack 3 系统和 Windows DDK 3790.1830 驱动编程环境中进行了工程实践验证.

### 2 APIC

当 I/O 设备产生中断时, 中断源首先在中断控制器中进行排队, 由中断控制器产生最后发往处理器的中断消息. 传统的中断控制器一般是 2 个 8259A 中断控制器级联, 形成一个可编程中断控制器(PIC). 其中一片的第二个端口和另一个级连, 将中断服务请求(IRQ)对应成数字序号 0-15. 为了有效地支持管理更多的外设中断和支持多核处理器, 新的硬件中断系统一般都采用了高级可编程中断控制器(APIC). APIC 可以提供 24 个中断接入端, 能够将中断发送到指定的处理器核上, 并对原有的 PIC 保持兼容. APIC 由一个 I/O APIC、多个 LOCAL APIC 和 APIC 总线组成. 具体可参照参考

收稿日期: 2012-09-28; 修回日期: 2012-11-23

基金项目: 国家基础预研项目(2011AC100001C100001)

文献[3]中给出的 APIC 与处理器连接的结构示意图。

## 2. I/O APIC

I/O APIC 位于处理器外面,它的基本功能是接收外设的中断信号,把请求中断的外设关联成相应的中断消息,传递给 LOCAL APIC。通过设置 I/O APIC 内部的 64 位中断重定向寄存器可以对中断消息进行配置,使 24 个中断输入工作在 APIC 总线上。中断重定向寄存器可以设置中断是电平或边沿触发、中断向量和优先级,中断目标处理器等。寄存器中的 IOREGSEL 和 IOWIN 两个寄存器被定位到内存空间,通过对这两个内存空间的访问可以实现对 IO APIC 其它寄存器的读写。根据文献[4],IOREGSEL 寄存器被映射到地址空间 FEC0xy00h, IOWIN 被映射到 FEC0xy10h。在实际使用过程中,首先用 IOREGSEL 选择需要访问的寄存器,然后对 IOWIN 访问即可读写相应的寄存器值。

## 2.2 LOCAL APIC

每一个 LOCAL APIC 都包含一组 APIC 寄存器,这些寄存器对应来自不同的中断:来自本地中断、来自 I/O APIC 的外设中断、处理器间中断、LOCAL APIC 内定时器中断、内部错误中断、性能监视器中断和温度监视器的中断。LOCAL APIC 对外连出两个引脚 LINT0 和 LINT1,对应本地中断和来自 I/O APIC 的外设中断;其他中断在内部完成。LOCAL APIC 的寄存器组对应到物理内存地址空间,通过读写这些地址的内存可以实现 LOCAL APIC 的配置。根据参考文献[3],APIC 寄存器被映射到物理地址空间为 0xFEE00000h 为起始地址的 4kbyte 字节区域,本文中需要用到的定时器寄存器被对应到地址 0xFEE00320h。

## 3 Windows 中断例程

### 3.1 中断预处理

当硬件中断发生时,处理器进入中断模式,处理器会自动将 Error Code(错误代码)、EIP(程序指针)、CS(段寄存器)、EFLAGS(标志寄存器)压栈,如果发生特权级切换的话,还需要压入 SS(段选择子)和 ESP(堆栈寄存器)<sup>[5-6]</sup>。接下来,处理器会根据由 APIC 传来的中断消息选择相应的中断服务。对于来自外设的中断,中断消息位于 I/O APIC 的中断重定向寄存器内。由于 APIC 支持 24 个不同的物理中断源,因此共有 24 个中断重定向寄存器,每个寄存器 64 位,详细信息见文献[3]。其中的[0:7]位即中断向量,对应本中断在 IDT(interrupt de-

scriptor table,中断描述表)内的偏移。通过中断向量表中的信息,可以获得中断服务程序的首地址。

IDT 是一张位于物理内存中的线性结构,共有 256 个单元,映射了 256 个中断、异常处理函数。操作系统在启动时,填充了 IDT 表,并将基地址存储到 IDTR(interrupt descriptor table register,中断向量描述表寄存器)。IDT 的每个表项实体 64 位。在微软的 WRK 源代码中,可以在 i386.h 头文件中查找到 IDT 数据结构如下:

```
typedef struct _KIDTENTRY
{
    USHORT Offset;
    USHORT Selector;
    USHORT Access;
    USHORT ExtenderOffset;
}KIDTENTRY;
```

IDT 中的实体是硬件与软件交接中断或异常的关口,被形象地称为门,共有中断门,任务门,陷阱门三类。本文中,我们关心其中的中断门。在该结构中,Offset 是处理例程入口的段内偏移低 16 位;Selector 作为段选择子,用于选择一个段描述符;Access[0:7]没有使用,命名为 reserved;Access[8:11]代表类型,命名为 type;Access[12]在中断门中总是为 0,命名为 always0;Access[13:14]代表描述符优先级,用于优先级控制,命名为 dpl;Access[15]是段存在标志,单独一位命名为 present;ExtenderOffset 为处理例程入口的段内偏移高 16 位。通过 IDT 中的信息,可以查得位于 GDT(Global Descriptor Table,全局描述符表)中的程序段信息,最后获得 CS:OFFSET 形式的的中断服务程序首地址,置入 PC,开始中断服务。

### 3.2 中断驱动程序

Windows 操作系统提供了一种被称为中断对象的数据结构来管理中断服务例程。中断对象创建时,操作系统从中断服务模板中拷贝中断分派代码完成它的初始化,当中断发生时,首先执行这些代码,接着执行设备驱动提供的 ISR(Interrupt Service Routine,中断服务例程)。在设备的驱动程序中,通过调用内核函数 IoConnectInterrupt 将驱动程序提供的 ISR 连接到指定 IDT 内的中断向量;卸载时通过调用内核函数 IoDisconnectInterrupt 解除连接。

### 3.3 中断优先级

#### 3.3.1 中断向量的优先级

APIC 定义了不同中断的优先级,除了被保留几个中断向量之外,外设可以使用剩下的中断向量

号. 对于 16—255 的不同的中断向量号隐含了不同的中断优先级. 它们的关系如下:

$$\text{Priority} = \text{Vector} / 16$$

式中, 16 最高, 1 最低. 每个中断优先级包含了 16 个中断向量, 这些向量又根据其中断向量低 4 位对应到细分的不同的优先级. 0 最低, 15 最高. 在每个 LOCAL APIC 内, 有一个任务优先权寄存器 TPR (Task Priority Register), 用于保存当前中断优先级. 如果 TPR 内的值为 0XYH, 那么等于或低于 0XYH 中断向量号的中断都被屏蔽. 事实上, 只要写入 0X0H, 即可将 0XF 以下的中断屏蔽.

### 3.3.2 Windows 中断的优先级

Windows 内部定义了一套优先级方案, 称为中断请求级别 (Interrupt Request Level, IRQL). 表 1 列出了 Windows 的 IRQL 方案<sup>[7]</sup>. 在该表格方案中, 4—26 号对应不同的硬件中断请求级. 硬件中断发生时, 在对应的硬件 IRQL 上执行中断服务程序. 事实上, 一些对于当前任务不是紧急的事情放到延时过程调用 (DPC) 中处理, 这样能够减少处理器停留在高 IRQL 上的时间, 从而快速响应低优先级中断. DPC 的引入, 是 Windows 非实时的原因之一.

表 1 Windows 中断请求级<sup>[2]</sup>

|     |                |         |
|-----|----------------|---------|
| 31  | HIGH_LEVEL     |         |
| 30  | POWER_LEVEL    |         |
| 29  | IPI_LEVEL      |         |
| 28  | CLOCK_LEVEL    |         |
| 27  | PROFILE_LEVEL  | 硬件中断    |
| 26  | DIRQL          |         |
| ... | ...            |         |
| 3   | DIRQL          |         |
| 2   | DISPATCH_LEVEL | 调度或 DPC |
| 1   | APC_LEVEL      | APC     |
| 0   | PASS_LEVEL     | 普通程序执行  |

### 3.3.3 IRQ 优先级和 IRQL 之间的对应

根据参考文献<sup>[2]</sup>中所述, Windows 通过内核数组 \_HalpVectorToIRQL 将中断向量对应成 IRQL, 同理, 利用 \_HalpIRQLToTPR 将 IRQL 对应成 TPR, 根据 TPR, 可以得到中断向量.

## 4 实时时钟的设计与实现

### 4.1 启用 LOCAL APIC 时钟

利用文献<sup>[1]</sup>中的方法, 启用 LOCAL APIC. 要完成这一目标, 需要对 LOCAL APIC 的寄存器进行写控制字. 整个过程涉及下列基本操作:

(1) 对 LVT 寄存器写控制字, 使能 LOCAL APIC, 并配置周期中断等工作模式.

(2) 对分频寄存器写预分频.

(3) 设置 Local APIC 定时器的初始计数寄存器.

(4) 设置 Local APIC 定时器的当前计数寄存器.

### 4.2 设置优先级

利用 3.3 节中介绍的中断优先级, 启用 Windows 中没有使用的中断向量. 中断向量 0XE2 的优先级高于系统时钟中断 0XD1, 因此新的时钟中断可以抢占系统时钟中断, 相应地, 对应的 IRQL 使用 1D.

### 4.3 时钟中断服务程序

时钟中断服务程序可以分为两个阶段, 先是根据中断发生时的现场执行相应的现场保护操作, 再是利用 HAL 中的函数执行中断服务程序. 利用参考文献<sup>[8]</sup>中的介绍和实验验证, 现给出新的时钟中断服务程序流程:

(1) 宏展开保护中断现场.

(2) 调用 HalBeginSystemInterrupt 准备执行中断程序.

(3) 关中断.

(4) 调用指定的中断操作 (本文中是 dispatch 函数).

(5) 开中断.

(6) 调用 HalEndSystemInterrupt 结束执行中断程序.

(7) 调用 Kei386EoiHelper, 中断结束.

### 4.4 中断服务程序注册

中断服务程序完成后, 需要向 IDT 注册, 使得系统能够在每次硬件中断到来时, 转去执行中断服务程序. 根据 3.1 节中对 IDT 的介绍, 结合 4.3 节中使用的 0XE2 中断向量, 只需要将中断服务函数的地址写到 IDT[E2] 中即可. 注册中断服务程序到 IDT[E2] 的关键代码如下<sup>[6]</sup>:

```
IDT [0xe2]. offset_low =
GetLow16_Of_32(Interrupt_Service);
IDT [0xe2]. offset_high =
GetHigh16_Of_32(Interrupt_Service);
```

## 5 数据测试和分析

### 5.1 实时性测试

参考文献<sup>[2]</sup>提出了利用 Intel 处理器内部的 64 位 TSC (Time-Stamp Counter, 时戳计数器), 来进行测试的方案. 在多核处理器下, TSC 的使用可

能会出现较大的偏差,因此微软推荐使用 Windows 提供的 API: KeQueryTimeIncrement 来测试. 该 API 能提供  $1\ \mu\text{s}$  的精度,可以满足本次实验测试需求. 本文设计测试方案如下:中断发生时,读取当前时戳,记录中断时时钟;当再次发生中断时,读取当前时戳,通过两次中断的间隔时间,可以得到中断服务程序执行的实际延迟时间. 在实际实验中采用的测试方法是统计间隔超过  $150\ \mu\text{s}$  的次数.

## 5.2 优先级测试

为了测试新时钟中断的优先性,设计如下方案:

(1)在原来的时钟中断上通过钩子函数,让其先延迟足够时间,保证在服务原时钟中断服务时有新的时钟中断出现.

(2)再次测试 LOCAL APIC 的时钟中断服务程序间隔,如果服务时间间隔还是在有效范围内,即认为抢占了原来的时钟中断.

## 5.3 数据分析和说明

按照 5.1 和 5.2 节中介绍的方法,测试在 60 s 中实时时钟超时的次数. 实验采用 intel E5300 双核处理器,测试针对 CPU0 和 CPU1 中的 CPU1. 通过对 APIC 编程,屏蔽通过 I/O APIC 发往 CPU1 的外设中断. 测试多组数据后,一组具有代表性的结果如表 2 所示.

表 2 测试数据表

| 测试项目 | 实时性测试  |          |          | 优先级测试  |          |          |
|------|--------|----------|----------|--------|----------|----------|
|      | 没有外部中断 |          | 有大量中断    | 没有外部中断 |          | 有大量中断    |
|      | CPU 空闲 | CPU 占用较高 | CPU 占用较高 | CPU 空闲 | CPU 占用较高 | CPU 占用较高 |
| 超时次数 | 0      | 0        | 3        | 0      | 0        | 4        |

实验数据测试表明,即使在 CPU 利用率很高的情况下,本方案仍然保持可靠地实时性,但是,当 I/O 操作频繁时,实时性将不能得到保证,如表中所示的在 60 000 次中断中出现了 3 次和 4 次超时. 数据表中的优先级测试数据表明,新的实时时钟中断能够抢占 Windows 原有的系统时钟.

经过对 Windows 的中断过程分析,认为以下这种情况的出现将会导致非实时:还差  $5\ \mu\text{s}$  触发 LOCAL APIC 的时钟中断时,Windows 内部的某个操作执行了屏蔽中断的命令,而该操作需要很长的延时,这将导致 LOCAL APIC 的硬件时钟中断一直被屏蔽. 除此以外,由于 Windows 内核的不可知性,不排除有其他导致非实时的原因.

由于不能保证在任何情况下实时时钟都能达到实时要求,因此:本文讨论和实现的实时时钟方案在大量中断的情况下会出现非实时的情况,这个问题导致这种方法具有一定的局限性.

## 6 WINDOWS 实时拓展方案

利用本文中给出的实时时钟,可以完成一个独立于 Windows、对指定实时任务提供实时保证的内核. 为了实现实时任务的保证,需要在各个层次上协同工作. 在硬件层,根据当前普遍使用的多核处理器,在第一个核上运行普通的程序,通过设置线程亲缘性把实时任务设置到第二个核上运行,即在多核处理器上专门设置一个或多个核用于执行实时任务. 在驱动层,需要提供实时的设备驱动程序,比如网卡驱动程序. 在驱动层以上的内核层,需要保证实时任务常驻内存,避免发生页替换时的巨大延迟. 除此以外,还需要一个任务调度器用于支持对多个不同优先级任务的实时调度. 在内核层和用户层之间,向内核层提供一系列的 API,用户层的编程人员调用这些 API 创建的应用程序即可以得到实时保证. 实时拓展方案结构图如图 1 所示.

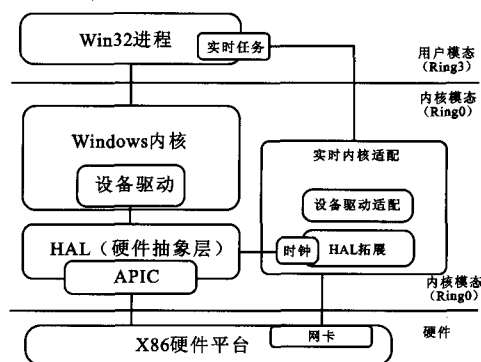


图 1 WINDOWS 实时拓展方案示意图

## 7 结束语

本文采用了对 APIC 编程的方法拓展了 WINDOWS 的时钟系统,设置了一个最小中断间隔  $100\ \mu\text{s}$  级别的时钟系统,并对其配置了中断服务程序. 在保证不被其他中断干扰的情况下,利用新的时钟系统可以保证每  $100\ \mu\text{s}$  引起一次拓展内核的重新调度,即新的实时系统每  $100\ \mu\text{s}$  调度一次,有效支持了特定的实时应用程序. 由于采用独立于 Windows 内核的实时拓展方案,可以保证 Windows 的完整性. 这种对实时拓展方案在源码开放的 Linux 操作系统上能有更好的实践发展,可以展开相应工作.

(下转第 123 页)

提出的高精度定时机制在进行定时误差优化以后能够达到很高的定时精度(图中定时误差不超过 400 ns)能够很好地满足实时调度的定时要求。

其次,验证处理器资源重新分配以后实时任务的执行情况可以看出在对 CPU 资源重新分配以后,实时任务的执行时间相对于处理器资源未分配时有了较大的差距,表明了对 CPU 资源重新分配能有效提高 Linux 系统实时性。

最后,验证方案中对系统外部中断进行指定分发处理以后实时任务的执行情况可以看出对外部中断指定分发以后,实时任务的执行时间相对于外部中断未指定分发时有了较大的差距,表明了对外部中断进行指定分发能有效提高 Linux 系统实时性。

## 7 结束语

本文从制约 Linux 系统实时性因素出发,从提高时钟精度、对处理器资源进行重新分配以及对中断进行指定分发三个方面来提高 Linux 系统的实时性。通过实验分别验证了此方案中所做出的改进对 Linux 的实时性有显著性的提高。以此方案为基础在实现一个更加简单的实时调度器运行在实时 CPU 核,采用合适的实时调度算法,并且再以内存映射机制为基础实现一套实时化内存管理机制就能实现一个完善的基于多核 PC 的 Linux 实时系统,这将是进一步研究的目标。

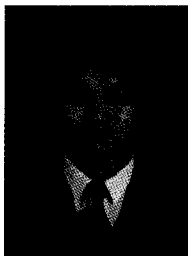
## 参考文献:

- [1] Daniel P Bovet, Marco Cesati. 深入理解 Linux 内核

[M]. 3 版. 北京:中国电力出版社,2007.

- [2] Robert Love Linux 内核设计与实现[M]. 2 版. 北京:机械工业出版社,2006.
- [3] Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman. Linux 设备驱动程序[M]. 3 版. 北京:中国电力出版社,2006.
- [4] 秋铁,周玉,邓莹莹. Linux 内核 API 完全参考手册[M]. 北京:机械工业出版社,2011.
- [5] 李小群,赵慧斌,叶以民,等. 一种基于时钟粒度细化的 Linux 实时化方案[D]. 北京:中国科学院软件研究所,2003.
- [6] 赖娟. Linux 内核分析及实时性改造[D]. 成都:电子科技大学,2007.
- [7] Vaidyanathan Srinivasan, Gautham R Shenoy, Srivatsa Vaddagiri, et al. energy-aware task and interrupt management in Linux[C]//proceedings of the Linux Symposium. Ottawa, ON, Canada: Linux Symposium, 2008:187-198.
- [8] Intel. Intel Corporation IA-32 Intel Architecture Software Developer's Manual Volume 3: System Programming Guide[M]. USA: Intel Corporation, 2006.

## 作者简介:



刘胜男, (1987—), 硕士研究生. 研究方向操作系统内核实时化改造、嵌入式计算及其应用。

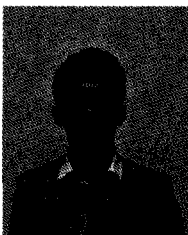
(上接第 119 页)

## 参考文献:

- [1] 潘汉. 基于 Windows 2000 的实时扩展研究[D]. 西安:西安电子科技大学, 2009.
- [2] 廖小勇. 基于 PC/Windows 环境的实时系统研究与实现技术[D]. 上海:同济大学, 2002.
- [3] Intel. Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1[M]. US: Intel Corporation, 2006.
- [4] Intel. 82093AA I/O Advanced Programmable Interrupt Controller (IOAPIC)[M]. US: Intel Corporation, 1996.
- [5] 张银奎. 软件调试[M]. 北京:电子工业出版社, 2008.
- [6] 谭文, 杨潇, 邵坚磊. 寒江独钓: Windows 内核安全编程[M]. 北京:电子工业出版社, 2009.

- [7] 张帆, 史彩成. Windows 驱动开发技术详解[M]. 北京:电子工业出版社, 2008.
- [8] 毛德操. Windows 内核情景分析—采用开源代码 ReactOS[M]. 北京:电子工业出版社, 2010.

## 作者简介:



蒋善锋男, (1989—), 硕士研究生. 研究方向为操作系统内核实时化改造、嵌入式计算及其应用。