第33卷 第10期 计 算 机 工 程 2007年5月
Vol.33 No.10 Computer Engineering May 2007
·软件技术与数据库· 文章编号：1000—3428(2007)10—0064—02 文献标识码：A 中图分类号：TP311

Linux实时性研究及其中断进程化的实现

洪雪玉，张 凌，陈宝钢，许宪成
(华南理工大学广东省计算机网络重点实验室，广州 510640)

摘 要：Linux 属于通用的分时操作系统，因此将它应用于实时系统领域必然存在一些不确定性问题，如内核不可抢占、时钟粒度粗糙、缺乏有效的实时调度策略等。为了解决这些问题，出现了一些如 RT-Linux、Kurt-Linux、Hardhat-Linux 等 Linux 实时性研究项目。文章在对 Linux 实时性研究的基础上，介绍了中断进程化的研究工作和实验结果。结果表明这种改进是显著的，较大地减少了内核的不确定延迟。
关键词：实时系统；Linux；中断进程化实现；确定性

Research on Real-time Linux and Implementation of Interrupt in Process

HONG Xueyu, ZHANG Ling, CHEN Baogang, XU Xiancheng
(Guangdong Key Laboratory of Computer Network, South China University of Technology, Guangzhou 510640)

【Abstract】The Linux kernel techniques, such as non-interruptible kernel, coarse-grained time mechanism and lack of effective real-time scheduler etc., can't well support the confirmability which is the basic characteristic of real-time operating system. Much research has been done on the real-time Linux in the world. In this paper, the shortcomings of Linux kernel when applied to real-time system and the relevant solutions implemented in some famous Linux projects, including RT-Linux, Kurt-Linux, and Hardhat-Linux, are introduced. Research on implementing interrupt in process is introduced and the final experiment shows that the research is effective and significant.
【Key words】Real-time system; Linux; Implementation of interrupt in process; Confirmability

1 概述

随着计算机技术的发展以及可使用资源的日益增多，实时系统的复杂性不断增加。这要求操作系统不仅要提供丰富的功能，还要提供较好的实时性能。传统的实时操作系统主要应用于工业控制、航天等硬实时领域。它们具有很好的实时性能，但功能较单一。而通用操作系统具备丰富功能，但不能提供较好的实时性能。为了解决这个问题，业内提出了很多解决方案，而增强通用操作系统的实时性是主要方法之一。凭借强大的网络功能、丰富的系统功能以及可裁减定制等特点，Linux 已成为嵌入式实时领域最重要的操作系统之一。

Linux最初的设计目标是分时操作系统，因此将它应用于实时系统需要解决下列问题：(1)频繁关中断将导致中断丢失，导致由中断触发的实时任务不能被立即调度执行。(2)为了减少关中时间，Linux将中断处理程序一分为二，即硬中断(top half)和软中断 (bottom half)。前者必须关中运行，后者是由前者调度的中断服务程序部分，可以开中运行。但是，无论是硬中断还是软中断，它们的执行都没有相应的调度机制，不能在统一的调度框架下，与系统中的其它任务进行资源竞争。因而它们的执行具有不确定性，这与实时系统要求可确定性的特点是相违背的。(3)粗糙的时钟粒度不能够提供精确的定时以满足实时应用微妙级的响应需求。(4)Linux内核的不可抢占性(2.5.4 版本之前)将产生优先级反转(priority inversion)问题。(5)缺乏有效的实时任务调度机制和调度算法。(6)虚拟内存管理等技术产生的系统延迟和不确定性。在实时系统中，这些不确定性是不能容忍的[1]。

因此，为了实现 Linux 对于实时应用更好的支持，除了需要提供一个细粒度的时钟管理机制外，还需要尽量避免或降低内核中的一些不确定因素。

2 Linux 实时性改造的国内外研究情况

许多国内外大学和研究机构均开展了对Linux实时性的研究和改造，如RT-Linux[2]通过软件模拟中断管理器的机制解决Linux频繁关中断造成中断丢失。Kurt-Linux[3,4]采用UTIME所使用的提高Linux系统中的时钟精度的方法实现了微秒级的实时精度。Hardhat Linux实现了内核的可抢占性。

目前，国内的一些大学和研究院校也开展了 Linux 实时性的研究工作。但其中的核心技术主要参考了上述的实时 Linux 项目，如中科院的 RFRTOS，就借鉴了 RT-Linux 的软断模拟中断控制器的技术、Kurt-Linux 的时钟管理机制、RED-Linux 的实时调度框架等。

3 中断进程化的研究与实现

前文总结了 Linux 在实时应用方面存在的一些问题，并介绍了几个典型的实时 Linux 研究项目在解决上述问题时所采用的技术方案。但对于问题(2)的研究仍不多见。对问题(2)的研究是基于这样的考虑：在 Linux 实现中，中断具有最高的执行优先权。当有中断到达时，CPU 将暂停当前任务，转入中断处理程序的执行。这种实现机制将可能导致下列问题：(1)由于中断处理程序没有相应的进程上下文，主要以内核函

作者简介：洪雪玉(1977－)，女，博士生，主研方向：操作系统，嵌入式系统，多媒体技术，计算机网络；张 凌，博导；陈宝钢、许宪成，博士生
收稿日期：2006-06-27 **E-mail：**xyhong@scut.edu.cn

数方式实现，因而其执行时间不可控。而如果被暂停的任务具有较高的实时性要求，那么它的执行将受到这种不确定延迟的影响，而实时系统最主要的特点就是确定性。(2)中断对应的处理程序的重要程度和实时性要求不一定高于被暂停的任务，因此导致了“优先级反转”的问题。因此，有必要研究一种机制以降低原系统中断机制导致的大量内核不确定延迟。文章采用了中断进程化的解决方法，以期达到两个目标：1)减少中断产生的内核不确定延迟；2)使得中断的执行具有相应的进程体。

有关中断进程化已经有了一些相关的研究，比如中科院的RFRTOS[5]就实现了中断进程化，RFRTOS基于Linux 2.2 内核之上。根据文献[5]的测试结论，由于多种原因，硬中断的进程化不但增加了中断延迟，而且软中断的进程化所产生的内核延迟也没有明显的减少。本文对于中断的进程化研究基于Linux 2.4，分别对硬中断和软中断进行了进程化，实验结果表明，硬中断具有一定的改进，软中断具有显著改进。

3.1 硬中断进程化及其实验结果

实验目的：将原中断的硬中断以进程的方式实现，以期减少内核的不确定延迟或减少关中时间，硬中断只需“唤醒”相应的进程，功能代码实现如下：

```
on each IRQ reaching
do
wake_up ( IRQ_Process[irqnum] )   //唤醒中断号为 irqnum 的中
//断对应的处理进程
done
```

其中，对 wake_up (int irqnum)进行了优化。

对网络硬中断进程化之前与之后的内核延迟做了比较。实验环境搭建如下：在测试主机上启动 aparche 服务器，然后通过 3 种不同的用户并发访问数来观察测试主机在不同网络负载下的内核延迟变化。采用的测试工具是 flood。记用户并发访问数为 user_num，每个用户对测试主机发送的请求数为 req_num，进程化之前每个网络硬中断造成的平均不确定内核延迟为 D1，总内核延迟记为 T1，其中：

T1 = D1 * user_num * req_num

进程化之后每个网络硬中断造成的平均不确定内核延迟为 D2，总内核延迟记为 T2，其中：

T2 = D2 * user_num * req_num

实验结果见表 1。

表 1 网络硬中断进程化之前与之后的内核延迟（μs）

User_num	Req_num	D1	T1	D2	T2
20	1 000	10.217	10 217	2.565	2 565
50	1 000	10.363	19 363	2.462	2 462
100	2 000	9.746	9 746	2.689	2 689

根据表 1 可知，硬中断进程化之后，平均不确定内核延迟有了一定的改进，大约从 10μs 减少到 2.5μs，总内核延迟大约从 10ms 减少到 2.5ms。因此，随着网络业务量的增加，即 user_nu 和 req_num 两个因子的增加，T2 将大大小于 T1，这对于侧重于网络访问功能的实时系统具有重要意义。

3.2 软中断进程化及其实验结果

中断的大部分处理是在软中断部分完成的，而硬中断只是完成一些必须立即处理的任务。如网络中断的硬中断只需将到达的数据包保存到内核缓冲区，对数据包的具体处理由软中断完成。因此，软中断一般需要比较多的处理时间，产生较多的内核延迟。如果软中断进程化能够显著降低内核的不确定延迟，将具有重要意义。

系统软中断功能模型描述如下：记 do_softirq 为软中断执行模块，softirq_queue 为软中断处理队列，ksoftirq_task 为执行软中断处理的内核守护进程。原 Linux 系统的软中断模型 I 实现如下：

```
begin   do_softirq
    for   each   softirq_queue[i] //遍历队列
    do
        execute    softirq_queue[i]
    done
    wake_up (ksoftirq_task ) //执行新的软中断队列
end   do_softirq
begin   ksoftirq_task
     other   pre-processing
     do_softirq    //  主要执行体
     other   post-processing
end   ksoftirq_task
```

改进后的软中断模型 II 实现如下：

```
begin   do_softirq
    wake_up (ksoftirq_task ) //唤醒内核守护进程，并由它负责所
//有的软中断队列处理
end   do_softirq
begin   ksoftirq_task
    for   each   softirq_queue[i] //遍历队列
    do
        execute    softirq_queue[i]
    done
end   ksoftirq_task
```

根据上述功能模型可知，原 Linux 系统软中断处理队列主要在内核函数中执行，这样将产生大量的内核不确定延迟。改进后的软中断处理模型将所有的执行队列交给内核守护进程处理。由于进程本身是可调度的，可以在一定的调度机制下参与系统调度，具有可预见性，因此可以大大减少内核不确定延迟，其不确定延迟将由模型 I 中执行 do_softirq 模块所需的时间降低到模型 II 中唤醒守护进程所需的时间。

下面对模型 I 与模型 II 产生的内核不确定延迟进行比较。分“系统轻度负载”与“中度负载”两种情况进行测试。“中度负载”是在“轻度负载”的基础上运行一个读/写磁盘的程序，该程序实现了不断将一个文件内容写到另一个文件中。每种负载情况进行 5 次测试，每次测试共进行 200 次软中断的执行，并分别取得其最小值、最大值和平均值。

系统在轻度负载情况下的测试结果见表 2。系统在中度负载情况下的测试结果见表 3。根据测试结果，在轻度负载下，内核不确定延迟由软中断进程化之前的 4.084μs 减少到软中断进程化之后的 0.018μs；在中度负载下，由 4.836μs 减少到 0.204μs。上述两种情况均使得内核不确定延迟由微秒级降低到了纳秒级。记软中断造成的平均不确定内核延迟为 softInt_ kdelay，系统软中断总数为 softInt_num，总内核延迟为 softInt_total，其中 softInt_total = softInt_kdelay * softInt_num。系统中包括网络、磁盘等各种类型外部中断，而且频繁发生，一般情况下 softInt_num 很大，因此 softInt_ kdelay 的改进将使得 softInt_total 大大减少，总处理时间将发生数量级的改变。此改进对于实时系统具有重要意义。

(下转第 94 页)