维普资讯 http://www.cqvip.com

第30卷 第18期 计 算 机 工 程 2004年9月
Vol.30 №18 Computer Engineering September 2004
·软件技术与数据库· 文章编号：1000—3428(2004)18—0089—02 文献标识码：A 中图分类号：TP316.2

基于Intel XScale PXA255处理器的Linux实时系统设计

杜 旭，冯 缜
(华中科技大学电信系，武汉 430074)

摘 要：XScale PXA255是Intel推出的一款基于ARM核的高性能微处理器，其核心工作频率最高可达400MHz，目前各大Pocket PC厂商已推出了基于PXA255的产品。该文分析了现有ARM Linux内核的实时处理缺陷，并提出了一种基于此处理器的Linux实时系统实现方法。

关键词：实时系统；ARM；XScale PXA255

Design of Linux Real-time System Based on Intel XScale PXA255 Processor

DU Xu，FENG Zhen
(Dept. of Telecommunications,Huazhong University of Science & Technolgy,Wuhan 430074)

【Abstract】 XScale PXA255 is a high-performance ARM core based micro processor developed by Intel with maximum core frequency 400MHz.This paper analyzes the ARM Linux kernel limitation on real-time processing and gives a real-time Linux implementation based on XScale PXA255 processor.

【Key words】 Real time system；ARM；XScale PXA255

Linux作为一种稳定高效的开放源码式操作系统，在各个领域都得到了广泛的应用，然而在实时处理方面的性能却不尽如人意。Linux内核对ARM体系结构微处理器的支持已经比较成熟，而在实时性方面却没有太大的改进，仍然受到了Linux内核机制的约束。虽然Linux内核提供了有限的POSIX支持，可以通过系统调用指定一个进程为实时进程，并提供两种实时调度策略FIFO(先来先调度)和RR(轮换制调度)，但这两种策略只是简单地增加了实时进程的优先级，不能从根本上提高Linux的实时任务处理能力。

XScale PXA255与ARM V5TE指令集兼容，沿用了ARM的内存管理、中断处理等机制，并在此基础上做了一些扩展，如DMA控制器、LCD控制器等。我们首先对ARM平台的Linux内核进行分析，然后针对其实时处理能力不强的原因提出相应的改进办法。

1 ARM Linux内核实时处理方面的不足

在硬实时系统中，实时任务都有各自的时间要求，如周期性任务执行的时间间隔，任务的截至时间等。操作系统必须尽快地调度即将执行的任务，否则就会引起灾难性的后果。而现有Linux操作系统无法做到这一点，这是因为：

(1)缺乏高精度定时器，无法通过时钟中断精确地调度任务。ARM Linux内核时钟中断间隔为10ms,对于实时处理远远不够。

(2)标准Linux中实现了FIFO和RR两种固定优先级实时调度策略，对于有精确时间要求的任务还是不够的。

(3)Linux内核不可抢占，内核任务调度函数schedule( )，只能由进程在内核中主动地调用，或者进程从系统空间返回到用户空间的前夕被动的发生。而这种“被动”的调度也不是每次都能发生的，发生调度情况主要有如下几种：

1)在时钟中断服务程序中，发现当前进程连续运行的时间过长。

2)当唤醒一个睡眠中的进程时，发现被唤醒的进程比当前进程更有资格运行。

3)当一个进程通过系统调用改变调度政策或礼让。

(4)Linux内存管理模块使用虚拟内存来为用户进程提供较大的寻址空间，当系统中空闲页面数不够时，内核线程kswapd会把一些已经很久不用的页面交换到外部存储器上。这种机制在运行需要很大内存的应用程序时得到了充分的运用。而在嵌入式系统中一般没有外部存储介质，在实时系统中由于外部存储介质读写的不可预测性，必须禁止页面换入/换出机制。

2 对ARM Linux内核的修改

针对基于ARM核的Xscale PXA255处理器，必须根据如上要点对现有ARM Linux操作系统做出修改，以达到实时性要求。

2.1 根据Xscale PXA255处理器硬件特性实现高精度计时器

现有Linux时钟中断的间隔时间固定为10ms，如果简单地减小时钟中断间隔时间，则会加重系统的处理负担。在实时应用中，一些实时任务有精确的时间要求，可以依据实时任务的时限来产生时钟中断。例如任务TaskA在时刻t需要得到处理，只要在运行到时刻t能产生一个时钟中断，就可以通过内核调度TaskA执行。此外，高精度计时器是下一节阐述的EDF策略实现基础。

XscalePXA255处理器中的OST(Operation System Timer)模块，输入时钟为3.6864MHz，32位OSCR(OS Timer Count Register)，32位OSMRx(OS Timer Match Register x=0－3)，OIER(OS Timer Interrupt Enable Register)，在每个输入时钟上升沿OSCR的值加1，与OSMRx的值进行比较，如果相同且OIER中的相应的中断允许位为1(允许中断)的话，则产生

作者简介：杜 旭(1969－)，男，讲师、博士，研究方向：高速宽带IP网络技术及应用，多媒体技术等；冯 缜，硕士生

收稿日期：2003-07-31 **E-mail**：rosser@eyou.com

维普资讯 http://www.cqvip.com

一个中断。通过动态地对OSMR进行编程来控制中断产生的时间，并规定产生时钟中断的最大时间间隔为10ms,即至少10ms产生一次中断。

2.2 针对实时任务的时限要求提供EDF调度策略

实现了动态优先级的EDF(Earliest Deadline First)调度策略,把采用EDF策略的进程简称为EDF进程。

为了较精确地反映自系统启动以来经过的时间，在内核中引入系统时间标识(System time stamp),类似于X86 pentium体系结构中的TSC(Time Stamp Count)。这是一个64位的全局变量sys_ts，单位为一个OST时钟周期(1/3.6864MHz)。

```
union sys_ts {
        unsigned long long ts;
        unsigned long hlts[2];
} sys_ts;
```

由于OSCR的值为32位寄存器，在经过232/3.6864MHz = 1165.08s后，从0开始计数。在每个时钟中断产生后，读取OSCR的值，对系统时间进行更新。如果发现OSCR的值比上一次读取的值小，说明OSCR已经重新开始计数，需要进行进位。

```
if ( ( count = OSCR ) < sys_ts.hlts[0] )
 sys_ts.hlts[1]++;
 sys_ts.hlts[0] = count;
```

此外在进程控制块task_struct结构中加入一些时间参数如任务的起始时刻(deadline)，中止时刻(stop_time)等，并且修改内核调度函数schedule( )。

在内核中维护了两个队列，一个队列task_ready_queue，只有处于task_ready_queue中的进程才有机会占有CPU，schedule( )选择task_ready_queue中第一个进程来执行。另一个队列EDF_task_queue按照EDF进程的起始时间进行排列。如果系统中没有EDF进程，task_ready_queue按照进程的优先级来排列，EDF_task_queue为空。否则应在任务执行的时刻产生中断，时钟中断处理函数把EDF_task_queue中的EDF进程放到task_ready_queue的最前面，时钟中断结束后由schedule( )调度EDF进程运行。至此，时钟中断处理函数的功能如下：

(1)更新系统时间标识sys_ts；

(2)从EDF_task_queue中去除当前时刻需要处理的EDF进程，并加入到task_ready_queue；

(3)对OSMR编程以设置下一次时钟中断产生的时刻：若EDF_task_queue中的最先执行的任务deadline与当前时刻相差不到10ms,那么设置中断产生时刻为deadline,否则设置为当前时刻加10ms；

(4)调用do_timer( )函数，完成必要的内核操作，如递增内核变量jiffies,激活底半操作等。

为了让应用程序能够使用EDF调度策略，可以在内核中为用户空间提供相应的系统调用如下：

pid_t sys_create_EDF_task( ): 创建EDF进程,建立task_struct结构，包括建立其内核堆栈，信号处理函数表等。并将其加入EDF_task_queue中。

int sys_destory_EDF_task(pid_t pid): 结束EDF进程，并将其从EDF_task_queue中移除。

long long sys_get_time_stamp( ): 读取当前系统时间标识。

int Set_EDF_Task_Start_Time(pid_t pid, struct timeval* time): 设置任务的开始时间。

Set_EDF_Task_Stop_time(pid_t pid, struct timeval* time): 设置任务的终止时间。

2.3 提高Linux内核的可抢断性

一般来说，Linux内核处于以下状态时不能进行调度，否则可能会引起混乱：

(1)内核正在处理中断。只有中断处理完毕后，且返回用户空间才有可能产生调度。而返回内核空间时不能引起调度，这是由于Linux设计者们规定了内核不可抢占。

(2)进入了一块由Spin lock, read lock或write lock保护的区域。在对称多处理器(SMP)体系结构中，Linux使用这些回旋锁来防止对临界区域的重入。

(3)内核正在执行调度函数schedule( )。它在系统中选择一个最适合运行的进程执行。

在以上前提条件下，为提高内核的可抢断性并保证临界区域的不可重入，在内核中插入一些抢断点，使运行于临界区以外的进程，即使在内核中也能被抢断。对内核做出了如下修改：

(1)在进程控制块task_struct结构中增加lock_count成员，非0表示当前进程进入了临界区域，初始值为0

(2)进入由回旋锁Spin Lock保护的临界区域当前进程的lock_count加1，离开时lock_count减1，并判断lock_count的值，如果为0,则可进行一次调度，即临界区域的结束位置可作为一个抢断点。

(3)当进程在系统空间产生了一次中断，中断处理完毕返回的时候，同样判断lock_count的值，如果为0，则可进行调度。

(4)在内核中比较长的系统调用(主要集中在内存管理与文件系统模块)中插入一些抢断点，这样可以减小内核的抢断时间。

2.4 内存管理机制的修改

Xscale PXA255使用ARM核的内存管理机制，基于分段或者分页的管理。使用分段管理时，段的大小为1MB，也就是说最多有$2^{32}>>20 = 4k$个段。在分页管理中定义了3种页面：微页面页面大小为1kB，小页面页面大小为4kB，大页面页面大小为64kB。

可以使用系统调用swapoff来禁止页面换入换出，从而不需改变内存管理机制与其他内核模块，如文件系统之间的接口。

3 对系统性能的测试

在实现了上述修改后，使用GNU工具Lmbench对标准ARM Linux内核和改进过的实时内核进行了测试。LMbench是一套的微基准测试程序,可测试内核中内存读写的带宽以及一些标准POSIX调用延迟测量(由于内存读写带宽很大程度上依赖于硬件性能，在此只列举了延迟测量的结果)。

表1 测试结果(单位为μs)

	标准ARM linux内核	实时Linux内核
read 调用	0.938	0.635 3
write 调用	0.599 4	0.341 5
stat 调用	10.854 6	7.965 9
fstat 调用	1.694 4	1.156 3
open and close调用	20.709 8	14.366 1
对10个fd进行select操作	5.883 4	3.862 3
对100个fd进行select操作	39.625 0	27.235 7
对10个tcp fd进行select操作	6.893 2	4.893 1
对100个tcp fd进行select操作	55.152 2	40.398 1
信号处理函数(signal handler)的安装	2.161 3	1.679 2
信号处理程序卸销	3.007 3	2.083 1
管道(pipe)延迟	85.808 9	66.792 8
AF_UNIX域套节字延迟	146.234 7	127.569 7

(下转第98页)

大变化时，才会产生E(D)的更新操作，因此在执行更新操作时λ 的值不应太大；Jit'_D为上一小节中的计算数值，每当接收到一个语音分组时，该值都要进行递归迭代更新运算；β、α ∈ (0, 1) 的值用来确定稳态区间的范围，从而决定当前判据的最小延迟D_{min}=β D_{TH}和最大延迟D_{max}=α D_{TH}，其初始值分别为$β_0$和$α_0$，同时在对语音编码算法进行调整后，β、α 的值也要进行相应的更新，其更新幅值为0.05,以防止语音编码算法的过渡快速的变化。在状态S11进行语音编码的更改时，为了避免造成多个终端的同时操作，在调整之前应有一随机延迟。

4 试验结果及分析

试验环境：针对端对端的控制机制进行试验，试验环境的设计以能够方便分析、易于观察和测试为原则，因此选择了窄带网络链路进行试验。这样虽具有一定的特殊性，但简化了试验和分析的复杂性，并且算法的控制效果易于观察。

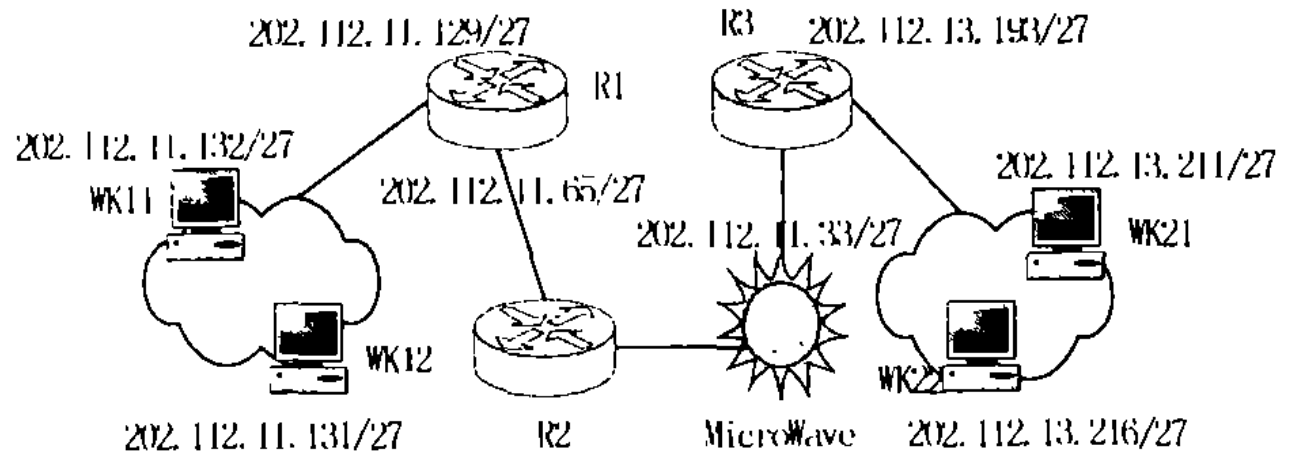


图5 试验环境

选用2Mbps的网络链路作为试验环境，如图5所示，本线路有两个子网202.112.13.192/27和202.112.11.128/27，路由器R1、R2、R3，和微波发射接收站组成，WK11、WK21、WK12和WK22为pentium4的主频为1.4Ghz的win2000工作站。

试验方案为：选用的试验链路为交大网络所和西北网络中心的一条备用线路，在校园网正常运行的情况下可以认为没有用户流量，因此首先要产生一个背景流量来模拟网络的正常使用。试验过程如下：

(1) 启动工作站W12和W22上的sniffer工具发包，产生1.5Mbps的双向背景流量。

(2) 在WK11和WK21两端分别启动通信系统程序VoipTH1进行G.711编码的RTP分组传输，同时利用一个简单的算法控制程序控制着在其后的1秒种内，随机启动10个收发线程，线程的发送速率分别模拟语音编码算法G.711和GSM G729的负载64kbps和32kbps。在VoipTH1的语音分组序列号为200时停止全部的发送线程。

(3) VoipTH1中不加载控制算法的情况下重复上述试验过程，在202.112.13.211端获得的分组统计效果如图6所示，其中虚线为终端在算法控制下的受控延迟线，实线为不加载控制算法时的自由延迟线，在两种情况下，D_{TH}=300ms，$β_0$=0.5, $α_0$=0.8, Φ =1/4，λ =0.5 。

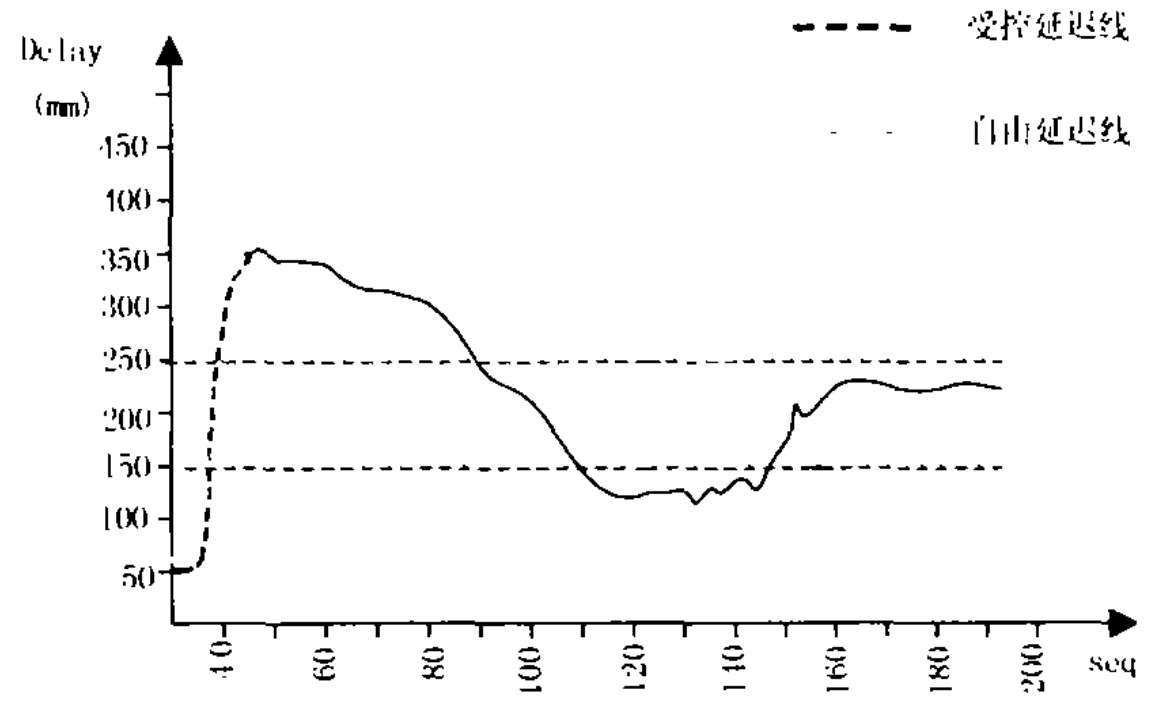


图6 分组延迟效果对比

试验分析：如图6所示，在VoipTH1启动时，语音分组的延迟在50ms以下，随着其它模拟线程的随机加入，网络的流量迅速接近链路的总带宽，导致分组的延迟快速增长至350ms左右，当延迟超过250ms时，E(D)+ Jit'_D大于区间的上限D_{max}，在两端控制算法的作用下，语音的编码算法由G.711转为GSM，网络中的总的通信流量减小，受控延迟曲线回落到120ms左右，这是受控延迟曲线的分组延迟小于150ms时，既E(D)+ JitiD小于区间的下限，则在控制算法的作用下，通信终端的由低速率语音编码算法恢复到高速率的算法，受控延迟曲线又上升至250ms左右，而自由延迟曲线则基本不变。

5 结束语

通过以上的理论分析和试验证明，基于抖动激励的终端延迟控制算法具有较好的延迟和拥塞控制效果。另外，当网络负载较轻，语音采样将以压缩率最小的算法进行编码，这时的分组延迟应明显的低于判据区间的下限，因此终端的控制算法将不起作用。

参考文献

1 ITU-T Recommendation H.323.Packet-based Multimedia Communications Systems(version4)[S].ITU Telecommunication Standardization Sector Study Group 16, 2000-11
2 糜正琨. IP网络电话技术[M]. 北京:人民邮电出版社, 2000-06
3 Scott Keagy著. 李真文译. 语音与数据集成网络[M].北京:电子工业出版社, 2002-01
4 Cray A. VOIP[J]. Data Communications, 1999, 28(9):45-48
5 朱 利,周俊辉.基于RTT的自适应拥塞控制研究[J]. 计算机学报, 2000,23(7):705-710

☆☆

(上接第90页)

可以看出经过高精度计时器改进，调度策略改进，内核可抢断性改进，内存管理改进后的ARM Linux系统性能有了较大的提高。

4 结束语

由于Xscale PXA255处理器的优良性能且耗电量低，已被多家PDA厂商用于其主打产品。本文针对Intel XScale PXA255处理器的硬件特性提出了一种对Linux内核进行改造扩充以提高实时性能的实现方法，为使用Linux操作系统的解决方案提供了实时性支持。

参考文献

1 张云生.实时控制系统软件设计及应用.北京:国防工业出版社,1998
2 Lin K J,Herkert A.Jitter Control in Time-triggered Systems.In proc. 29th Hawaii Conference on System Science,Maui,Hawaii,1996-01
3 Beck M.Linux Kernel Internals.Addison-wesley, 1998
4 Golatowski F,Hildebrandt J,Timmermann D.Scheduling Coprocessor for Enhanced Least-laxity-first Scheduling in Hard Real-time Systems. 11th Euromicro Conference on Real-time Systems,York,England,1999
5 Wang Y C, Lin K J.Implementing a General Real-time Scheduling Framework in the RED-Linux Real-time Kernel.In 20th IEEE Real-time Systems Symposium,Phoenix, Arizona,1999