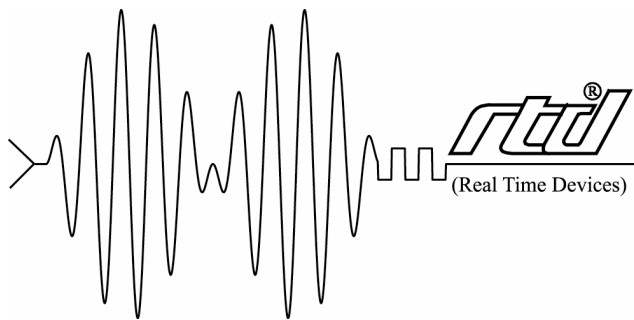


APPLICATION NOTE

Linux Interrupt Performance



RTD Embedded Technologies, Inc.

"Accessing the Analog World"®

SWM-64000021
rev A

APPLICATION NOTE

Linux Interrupt Performance



RTD Embedded Technologies, Inc.

103 Innovation Boulevard
State College, PA 16803-0906

Phone: +1-814-234-8087

FAX: +1-814-234-5218

E-mail

sales@rtd.com

techsupport@rtd.com

web site

<http://www.rtd.com>

Revision History

09/27/2005 Revision A issued

Published by:

RTD Embedded Technologies, Inc.
103 Innovation Boulevard
State College, PA 16803-0906

Copyright 2005 by RTD Embedded Technologies, Inc.
All rights reserved
Printed in U.S.A.

The RTD logo and cpuModule are registered trademarks of RTD Embedded Technologies. MS-DOS and Windows are registered trademarks of Microsoft Corporation. Linux is a registered trademark of Linus Torvalds. All other trademarks appearing in this document are the property of their respective owners.

Table of Contents

TABLE OF CONTENTS	4
INTRODUCTION	5
INTERRUPT PERFORMANCE	5
PROCESS SLEEPING AND WAKING	5
NUMBER OF PROCESSES IN SYSTEM	5
PAGE FAULTS	6
PROCESS SCHEDULING	6
INTERRUPTS FROM OTHER DEVICES	6
SIGNALS	6
GENERIC COMPUTING ENVIRONMENT	7
CPU ISSUES.....	7
GRAPHICAL ENVIRONMENT.....	7
MULTIPLE DEVICES ON A SINGLE BUS	7
GENERIC DEVICES	7
DEVICE DRIVER SOFTWARE	8
BACKGROUND SYSTEM ACTIVITY	8
HIGHER PRIORITY INTERRUPTS	8
CONCLUSION	8

Introduction

RTD designs its Linux drivers to be as efficient as possible. However, sometimes circumstances beyond reasonable control exist which affect interrupt functioning or which cause observable performance differences when comparing Linux to DOS, Windows, or other operating systems.

Interrupt performance problems may manifest themselves in many different ways. Examples of behavior you may see include but are not limited to:

- * missed or lost interrupts
- * inability to reach a board's maximum throughput rate
- * interrupts held off for long periods of time
- * board FIFOs fill up
- * an application struggles to keep up with the data being generated by the hardware
- * a board stops generating interrupts
- * lost communication data, for example Ethernet packets or CAN bus messages

This document discusses some of the issues related to interrupts, describes how these topics affect system activity, and provides some ideas regarding actions to take to reduce interference.

Interrupt Performance

Most drivers provide an interrupt notification feature which applications can use to receive interrupt status asynchronously. Once this interrupt happens, an application usually performs some sequence of actions and then waits for another interrupt. This paradigm by its nature involves both kernel and user level; the driver resides in kernel space and your application resides in user space. This model provides flexibility and reduces complexity but has limitations like all software. There are many things which can affect performance when the user application waits for an interrupt and then is notified of its occurrence by the driver; these issues also affect interrupt performance in general.

Process Sleeping and Waking

Putting a user process to sleep in the kernel to wait for the interrupt and then waking it up is expensive in terms of time needed. This delay interferes with the usable work a process can do and how fast it can do it. Additionally, when a process exits the kernel there may be other work for it to do (for example checking for and handling signals) before it resumes what it was doing in user space.

Number of Processes in System

Dependent upon the other runnable processes in a system, your application may not be selected to run on the CPU for some time after it is woken up by an interrupt. Ignoring other factors for the moment, the higher the number of runnable processes the more likely it is that your application will not be selected to run right away.

To address the number of runnable processes issue, conduct an audit of the system and determine which processes you do not need. For each such process, some of the changes you can make are to modify the system configuration such that the process is not started at boot, ensure that the capability's kernel module is not loaded, or disable the functionality at the hardware level. The smaller the number of processes the more likely it is that your application will be granted the CPU when it needs it.

Page Faults

If the user application incurs a page fault, that is accesses an address which is valid but not currently loaded into main memory, the process must be put to sleep until the kernel brings that memory page into main memory from some device much slower than memory. Your application can perform no useful work while it slumbers.

A process can lock its memory pages into main memory to prevent page faults. Not having to wait for a page fault to be serviced means that a process has more time available for useful work. Locking pages into memory during program initialization ensures that all pages are resident in memory, thus any access to them will not fault. There are some situations in which page faults can still occur but locking pages into memory greatly reduces the likelihood of them occurring. Please see the `mlockall(2)` man page for more information.

Process Scheduling

Process scheduling priority affects which process gets chosen to run on the CPU. If your application has a scheduling priority lower than most other processes, it will not be selected to run if at least one of the higher priority processes is ready to run. Furthermore, a runnable process with a scheduling priority lower than that of a running process cannot preempt that process and begin running on the CPU.

You can change a process' scheduling priority and class under Linux. A process' scheduling priority can be increased so that it will be chosen to run before other processes with lower priority. Changing a process' scheduling class can disable scheduling time quanta for it, which means that the process will not be rescheduled just because its CPU time slice has expired; this has no effect if the process voluntarily puts itself to sleep. You must exercise extreme caution when modifying scheduling parameters because a process can enter a state where it will be given preference over all other processes, thus deferring certain system activity. The system may even become unresponsive in some ways. Please see the `sched_setscheduler(2)` man page for more information.

Interrupts From Other Devices

Other interrupts may affect an application. A system's interrupt controller usually assigns a priority to each interrupt. Thus, a higher priority interrupt can interfere with the interrupt your application is interested in by keeping it pending until the higher priority interrupt is serviced. Worse, if several interrupts occur while one is being processed some of them may be lost.

If it's possible to assign a specific interrupt, pick one that has a higher priority than other interrupts. This will allow your device's interrupt to take precedence over other lower priority interrupts that you may not be as interested in. Again, extreme caution must be used because deferring certain interrupts may affect system activity. You can also remove unnecessary system boards, disable unused devices in the BIOS, or disable any interrupts they may use.

Signals

A signal delivered while waiting for an interrupt causes the process to wake up before the interrupt occurs. The kernel makes the process runnable in order to execute its signal handler and the interrupt wait returns an indication that a signal was delivered while blocked. The application must then retry the interrupt wait, at which time the kernel puts the process back to sleep. Please see the earlier comments regarding process sleep and wake up.

You have two options available if signals cause missed interrupts. First, do not use signals in your application if possible. Second, try to minimize signal usage to lessen their adverse effect if you must use signals.

Generic Computing Environment

Keep in mind that most Linux distributions provide generic computing environments rather than real-time capabilities. This means that extra measures, which can increase interrupt & scheduling determinism and reduce system latency, are not taken. For example, in a non real-time Linux kernel, device drivers may hold off other system interrupts until their interrupt processing completes. Known culprits currently include the IDE and real time clock drivers; many others may exist. Be aware that some solid state storage devices actually sit on the IDE bus and therefore use the IDE driver.

Consider using a real-time Linux kernel as it provides better determinism and reduces latency so that events are more predictable. Currently, RTD drivers do not support real-time kernels. Since full source code is provided, you are free to adapt any driver to a real-time kernel.

CPU Issues

CPU type and speed affect the ability to process interrupts. In general, an RTD cpuModule based upon either the VIA Eden or the Intel Celeron processor exhibits better interrupt performance than an RTD cpuModule based on the AMD Geode processor. Normally, a higher CPU clock speed means increased interrupt performance. Obviously, exceptions exist.

If you suspect interrupt problems related to the CPU, you can either move to a different CPU family, get a faster CPU, or both.

Graphical Environment

Interrupt throughput may suffer when running a graphical environment. This is due to the overhead imposed by the X server and the resources it uses.

Assuming you do not require a graphical environment, you can disable the X server to remove another source of potential problems.

Multiple Devices on a Single Bus

Multiple sources of activity on a single bus may cause problems. For example, using two ISA or PCI cards which often contend for the bus may increase interrupt latency.

Remove unnecessary cards from your system, disable unused devices in the BIOS, or disable their interrupts. This decreases the chances of several cards contending for system resources at the same time.

Generic Devices

Devices such as keyboards, mice, COM ports, and network cards exist in many systems. These devices are so commonplace that we often simply forget they are there. Yet, they can interfere with another device's interrupt performance by generating their own interrupts, transferring data, and consuming system resources.

As a first step, minimize usage of the device or devices in question. As stated previously, you can remove any unnecessary hardware from your system if you think it is causing problems. Disable the device in the BIOS (if applicable), rebuild your kernel without its driver, or ensure that its driver module is never loaded into the kernel.

Device Driver Software

RTD designs its drivers to provide as generic a development environment as possible to allow the associated hardware to be used in many different applications. It's simply not possible to design one piece of software so that it works efficiently for every conceivable use.

Since full source code is provided, you may rewrite the software as you see fit to maximize its utility and effectiveness for the application you intend.

Background System Activity

Linux has much activity going on in the background to support critical system services. At regular intervals, Linux flushes in-memory buffers containing data not yet written to backing storage out to that storage; the more data there is to write, the longer it takes. Daemons awaken to respond to system events; the more events that occur, the more times these daemons wake up and the more work there is to perform.

Some system activities can be tuned in terms of how often they occur. You can also disable daemon processes that are not required or disable system services if they are not being used. For more information, please consult a good system administration reference.

Higher Priority Interrupts

A system's interrupt controller prioritizes interrupts. As a consequence, a higher priority interrupt can preempt an interrupt handler servicing a lower priority interrupt. If your device interrupt is of lower priority than other system interrupts, its handler may take longer than expected to service the interrupt.

You can modify your device's interrupt handler to disable all system interrupts upon entry and enable them on exit. This allows the handler to execute without interference. This comes at a price though, namely that other system activity may be adversely affected if associated interrupts are disabled; be very careful about disabling all interrupts. Note that disabling interrupts will not prevent other drivers from blocking out your device's interrupt for long periods of time.

Conclusion

The delays inherent in any of the above situations affect how quickly an application can respond to an interrupt and how many interrupts can be processed in a given period of time. The application may miss or lose interrupts, especially if interrupts occur quickly enough. A board may enter an inconsistent state if interrupts are not acknowledged reasonably.

As a last resort, you can rewrite the driver's interrupt handler or the driver to implement whatever functionality you require. The upside is that doing so offers the most control over interrupt response. The downside is that 1) this approach requires kernel programming knowledge and 2) you may need to add extra driver and library code to fully interface with your application.

Ultimately, the issues discussed herein lie beyond the control of a software engineer designing a device driver for a particular piece of hardware. In essence, it is the price you pay for using Linux (or any other operating system).