Literature Survey for COMS W4995-2 Project Benjamin Ip

Introduction

This paper presents the review of several performance measurement methodologies used to benchmark the VxWorks and RTLinux real-time operating systems. While various approaches to achieve fair and accurate metric have been suggested, only effective methods should be chosen to assert the RTOS suitability of running RT applications. This survey begins with a brief overview of the two operating systems followed by the examination of different metric used to evaluate them, and concluded with the challenges face to achieve the project goal.

Background

VxWorks is by far the most widely adopted commercial RTOS in the embedded industry. It is developed from WindRiver with the intention to design an OS with fast, efficient, and deterministic context switching. The *Wind* micro-kernel can support preemptive and round robin scheduling policies and maintain unlimited number of tasks with a maximum of 256 priority levels. VxWorks is also well known for its rich tool chain and run time library that significantly reduce the amount of time for application development. Despite the comprehensive features from VxWorks, it bares a high premium for royalty fee.

Unlike Linux, RTLinux provides hard read-time capability. It has a hybrid kernel architecture with a small real-time kernel coexists with the Linux kernel running as the lowest priority task. This combination allows RTLinux to provide highly optimized, time-shared services in parallel with the real-time, predictable, and low-latency execution. Besides this unique feature, RTLinux is freely available to the public[1]. As more development tools are geared towards RTLinux, it will become a dominant player in the embedded market.

Performance Metric & Measurement Approach

This section describes some of the *traditional* performance metric that this project will use for evaluating the above two OSes.

Context Switch

Real-time systems are typically implemented with multiple asynchronous tasks of execution. Because of this characteristic, a RTOS scheduler is usually designed to support multitasking. During task scheduling, a context switch is needed to suspend one task and immediately resume the other. Therefore, it is important to minimize the average context switch latency. Levine [1] proposed three metric: the *Suspend-Resume*, *Yield*, and *Synchronized Suspend-Resume* tests to benchmark context switch time. The *Yield* test cannot be run on VxWorks because it does not support an immediate task yield without delaying for a non-zero time interval [1]. On the other hand, the yield function in RTLinux always causes a schedule recalculation of all runnable processes [1]. Even using the remaining two tests still requires taking multiple measurements to obtain reliable result.

Priority Inversion

Priority Inversion occurs when a high-priority task is blocked waiting for a low-priority task to release a resource required by the high priority task. Priority inversion must be eliminated and modern RTOSes often incorporate their own priority inversion protocols. Levine's [1] method on detecting and observing priority inversion is complicated. A straightforward way to create a priority inversion scenario is to have three tasks running at low, medium, and high priorities, with the low and high priority tasks competing for the same resource [4]. The time between the high priority task requesting the resource and the low priority task releasing the resource will approximately indicate the effectiveness of each of the protocol [4].

[1] RTLinux is distributed by Finite State Machine

Interrupt Latency
Interrupt Latency is defined as the sum of the interrupt blocking time during which the kernel cannot respond to the interrupt and the dispatching overhead due to context saving, determination of interrupt source, and invoking the interrupt handler. For a particular interrupt, the latency also includes the execution time of other nested interrupt handlers. Sihol [3] took both the analytical and empirical approaches for measuring interrupt latency. Although, he has spent much time on elaborating the differences between the two approaches, only brief amount of text is dedicated to illustrate the measurement process. Sun's [4] fully dedicated his article on describing how to measure latency in the Linux environment. Since RTLinux is a variant form of Linux, Sun's example must be tuned for measuring RTLinux interrupt latency. Little research has been done publicly to measure VxWorks interrupt latency. Nevertheless, a FAQ that maintained by WindRiver [5] presented latency figures performed on a different but limited hardware platforms. Real-Time Magazine [6] does provide test plan enumerating the procedures for latency measurement generic to examine both RTOSes.

Synchronization
Synchronization is required for a real-time system when it is executing multiple tasks and each of them try to share and access the same resources. Both VxWorks and RTLinux provide a full suite of synchronization methods, but using these libraries do incur penalty. Obenland's[7] technique to measure semaphore overhead include two independent tests for subtracting the semaphore system call overhead to calculate the net latency incurred during semaphore creation.

Inter-Process Communications
Modern real-time applications are constructed as a set of independent, cooperative tasks. While semaphores provide a high-speed mechanism for task synchronization and interlocking, often a higher-level mechanism is necessary to allow cooperating tasks to communicate with each other. Message queue is one of the many ways to provide abstraction of inter-task communications.. According to Obenland's [7] article, message queue overhead is the time between a task request sending a message and another task receiving that message. Obenland emphasized that prior to the test execution, the message queue must be created with no message pending and the receiving task must be blocked waiting for the message.

Timing Measurement
Measuring the above metric requires certain degree of resolution, accuracy, and granularity. Steward's article [2] provides an excellent tutorial on how to select different timing measurement methods. He and the author from Real-Time Magazine [6] pointed out that software analyzer tools that come with the RTOSes must be exercised carefully because these tools varies greatly in timing resolution. Steward also identified that a good software analyzer should provide means to measure small segment of code, time trace to show process execution time, and minimal measurement overhead. Steward, however, suggested that the best tool to obtain accurate measurement is to use a logic analyzer because it gives very fine resolution and least obtrusion on real-time code. There are two approaches to using a logic analyzer. One way is to hook up the analyzer probe to the CPU pins. Another way is to send a pattern of signals to an output port, which are read by the analyzer as events. Finally, Steward discussed other approaches such as using the *clock* function and the *Prof* /*Grof* profiling tool as other means for measurement; however, they offer less granularity than the methods aforementioned.

Operating System Overhead
The context switch overhead includes time for RTOS to perform scheduling. Often, the time for scheduling is a function of the number of tasks on the ready queue. Therefore, the effect of RTOS overhead for context switching and scheduling must be considered if accurate measurements are needed. Steward [2] illustrates a general but detailed example on how to minimize the overhead.

Challenge and Future Work

In the remainder of the semester, I'll try to address/solve the following issues/questions:

- Should different metric be measured with a system load or without any background task?

- Should test code be developed entirely myself or obtained from third party? Should I use tools that come with the RTOSes, or simply use logic analyzer and small test code written to achieve timing measurements? Choosing a particular effort will significantly affect development time.

For the rest of this semester, I'll dedicate more time on configuring the target and the host, and proceed to generate a thorough test plan for each of the test metric. Efforts spent on implementing test functions will vary greatly depending on the direction that I take as described earlier.

References

[1] D. Levine, S. Flores-Gaitan, C. D. Dill, and D. C. Schmidt, "Measuring OS Support for Real-Time CORBA ORBs", in 4th IEEE International Workshop on Object-oriented Real-Time Dependable Systems 00', Santa Babara, California, Jan. 27-29.

[2] D. Stewart, "Measuring Execution Time and Real-Time Performance", Embedded System Conference, Spring 2001

[3] V. Sohal, "How To Really Measure Real-Time", Embedded System Conference, Spring 2001

[4] Jun Sun, "Interrupt Latency", Monta Vista Software, http://www.mvista.com/realtime/latency/

[5] "VxWorks FAQ", http://www.xs4all.nl/~borkhuis/vxworks/vxworks.html

[6] Real Time magazine, "Evaluation Report Definition", http://www.realtime-info.be, March 1999

[7] K. Obenland, "Real-Time Performance of Standards Based Commercial Operating Systems"

Additional References

[7] Intelligraphics, "Performance Utilization and Performance Benchmarking in Embedded Systems", http://www.intelligraphics.com/articles/EmbeddedBenchmarking_article.html

[8] R. Appleton, "Understanding a Context Switch Benchmark", Linux Journal http://www2.linuxjournal.com/lj-issues/issue57/2941.html, Jan. 1997

[9] Victor Yodaiken, "The RTLunix Approach to Hard Real-Time", http://rtlinux.org/documents/papers/whitepaper.html, Oct. 1997

[10] V. Yodaiken, "The RTLinux Manifesto", http://rtlinux.org/documents/papers/rtmanifesto/rtlmanifesto.html

[11] M. Barabanov and V. Yodaiken, "Real-Time Linux", http://rtlinux.org/documents/papers/lj.pdf, Mar. 1996

[12] V. Yodaiken, "An Introduction to Real-Time Linux", http://www.rtlinux.org/documents/RTLinux.ppt

[13] P. Wilshire, "Installing RTLinux", http://rtlinux.org/documents/installation_june_2000.html, 2000

[14] T.C. Siering, "RTLinux Installation Tips", http://rtlinux.org/documents/installation_notes.html, 2000

[15] M. Barabanov, V. Yodaiken, and E. Hilton, "RTLinux FAQ", http://www.rtlinux.org/documents/faq.html, 2001

[16] R. Lehrbaum, “Using Linux in Embedded and Real Time Systems”, http://embedded.linuxjournal.com/resources/lj/lj75/3980.php

[17] A. Ivchenko, “Real-Time Linux”, Embedded System Programming Magazine, May 2001

[18] A. Ivchenko , “Application Code and RTLinux”, Embedded System Programming Magazine, Jun. 2001

[18] M. Bunnell, “Solving the Embeddded Linux Challenges”, http://www.lynuxworks.com

[19] D. Bovet and M. Cesati, *Understanding the Linux Kernel*. Sebastopol, CA: O’Reilly, 2001

[20] comp.os.linux.embedded

[21] WindRiver Systems Inc, *Tornado User’s Guide,* Alameda,CA: WindRiver Systems, Inc, 1999

[22] WindRiver Systems Inc, *VxWorks Programmer’s Guide*, Alameda,CA: WindRiver Systems, Inc, 1999

[23] WindRiver Systems Inc, *Tornado Training Workshop*, Alameda,CA: WindRiver Systems, Inc, 1999

[24] comp.os.vxworks