Towards Real-Time Enabled Microsoft Windows

Xiang (Alex) Feng
Microsoft Corporation
Redmond, WA 98052
xfeng@windows.microsoft.com

ABSTRACT

Many computer scientists recognize the adverse relationship between Microsoft Windows, a general purpose operating system, which by design does not support Real Time, a specific purpose feature. However, the boundary between a general-purpose system and a special-purpose feature has begun to blur and Microsoft and its partners have kept working on adding real-time services to Windows. In this paper, we will describe existing real-time solutions for Windows, the on-going projects for the next release of Windows and future trends lead by hardware evolution.

Categories and Subject Descriptors

D.4 [**Operating Systems**]: General—*Microsoft Windows*; D.4.7 [**Organization and Design**]: [Real-time systems and embedded systems]

General Terms

Design

Keywords

Operating Systems, Microsoft Windows, Real-Time Systems, Embedded Systems

1. INTRODUCTION

Microsoft Windows [1] is arguably the most successful commercial software in history. Since the first release of Windows NT 3.1 in July, 1993, there have been 700 million Windows users in the world. Windows has changed the way people work, communicate, entertain and live.

[1]By Windows, we are referring to the family of Windows NT, 2000, XP and the up-coming release Vista. Because Windows 95, 98, ME and CE have different code-bases, hence are excluded from our discussion.

EMSOFT'05, September 19–22, 2005, Jersey City, New Jersey, USA.

However, when we see Windows from the lens of a real-time researcher, real-time research has little impact on the Windows family. Because real-time system research has traditionally focused on special-purpose operating systems., most of the usage has been limited to health care, military, avionics and academic settings. Although the importance of these systems can not be underestimated, they are simply not as widely used as general-purpose operating systems.

The longstanding argument that that real-time features are by nature not for general-purpose systems must be questioned as the boundary between embedded systems and general-purpose systems continues to blur. For example, most people agree that a set-top box is a typical embedded device. However, with the release of Windows Media Edition TV, it is difficult to keep traditional definitions.

Soon, we will see the continuing force of integrating work, communication and entertainment capabilities of PCs. PCs have become the center of offices and they will also soon become the center of living rooms and possibly kitchens with increasing demand for integrated PC capabilities. When all these functionalities come together, performance and, to our real-time researchers' interest in particular, predictability will become more and more important. Take VoIP (Voice over IP) for example [8], a mandatory VoIP 911 bill was recently introduced to enforce VoIP vendors to provide 911 emergency services to customers. This is evidence that consumers are relying more and more on IT technology to deliver time-critical services.

At the same time, the academia also witnessed research trends moving from a specific domain to a more general-purpose one. Extensive research has been done from static clock-driven scheduling to a more adaptive event/priority-driven scheduling; among the priority-driven scheduling algorithms we have studied from static fixed priority to dynamic deadline-driven scheduling. A recent example introduces the concept of an open system [7]. In an open system, resources are shared by different classes of applications, some hard-real-time, others soft-real-time or even non-real-time. Furthermore, applications and resources can dynamically join and leave the system. An open system is no longer a special-purpose system.

With all these factors considered, the era of general-purpose real-time systems is finally coming. It will still be a slow progress and we won't see hard real-time enabled on Windows anytime soon, but the trend will consistently progress.

In this paper, I will first describe some background of Microsoft Windows in Section 2, next Section 3 will discuss the current efforts to make Windows (mainly NT embedded

and XP embedded) real-time. I will then in Section 4 show the on-going changes in Windows Vista. A significant trend will be discussed in Section 5 before the final conclusion.

2. BACKGROUND

Windows is designed to be a highly responsive, general-purpose operating system. While fast response is absolutely desirable, it does not necessarily meet the requirement of real-time. Real-Time is about predictability, which in the case of system responsiveness, is a deterministic response. Before we discuss how to make Windows real-time, let us look at the OS design and the scheduler in particular in order to understand why the current Windows is not a real-time operating system.

Like many other operating systems including Unix and Linux, Windows has two running modes: kernel mode and user mode. Kernel mode and user mode are separated to prevent user applications from modifying, either maliciously or by mistake, the OS kernel and other applications. An oversimplified view of Windows can be found at Figure 1[11].

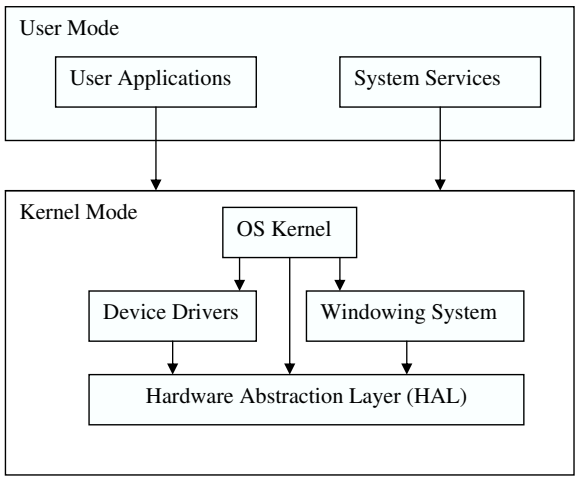

Figure 1: Oversimplified Architecture of Windows

The lowest layer of the OS is the hardware abstraction layer (HAL). Kernel and device drivers both run on the top of HAL in the kernel mode. In theory, windowing system should run in user mode because it is part of the system services. But due to performance reasons, it moved to the kernel mode as well. The user mode contains 1) user applications such as Internet Explorer and Office and 2) system services such as network services.

Windows kernel scheduler also takes a layered approach. Overall, the structure can be divided into two categories as shown in Figure 2: interrupts and threads. Interrupts include 1) hardware interrupts triggering interrupt service routine (ISR) such as interrupts from hardware devices, power fail interrupts and 2) software interrupts such as deferred procedure call (DPC). Interrupts always have higher priorities than threads, therefore the running thread will always be preempted whenever an interrupt arrives. Interrupt request levels (IRQLs) are used to coordinate among these interrupts, similar to the way priorities do to threads.

Thread priorities are used to schedule threads. As shown in Figure 3, Windows currently has 32 levels of priorities with 0 as the lowest and 31 as the highest. Interestingly

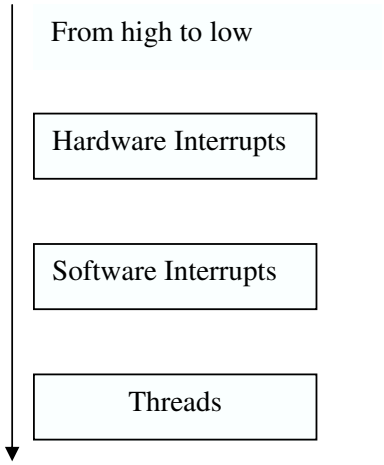

Figure 2: Kernel Scheduling Layers

enough, the highest 16 levels from (16-31) are called real-time levels. The name here is misleading because it doesn't mean that threads running on these levels will receive guaranteed CPU cycles. On the contrary and in the worst case, threads running on the highest priority level, namely priority 31, may receive no processor cycles because the processor may be busy with ISRs and DPCs, i.e., interrupts. Besides, system threads are not running on these levels. Setting an application thread on these levels may not necessarily help but possibly block system threads, and thus endanger the system stability. Due to these reasons, the real-time levels are not widely-used. If they have to be used, then extreme caution must be employed.

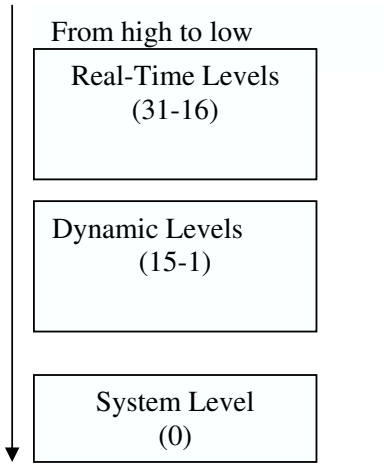

Figure 3: Thread Priority Levels

During implementation, the difference between real-time levels and lower levels (called dynamic levels) is that the scheduler will never change the priorities of threads running on real-time levels. For those threads running on lower levels, the scheduler will occasionally boost the priorities of some threads for a short duration simply to improve responsiveness for foreground applications or to avoid thread starvation.

Therefore, before running on a processor, a thread has to

first wait for hardware interrupts to finish, then the software interrupts, then the threads with higher priorities. When the thread finally starts to run, whenever an interrupt arrives, or a higher priority thread becomes ready, the thread will be preempted and have to wait for them to complete. The current kernel architecture is inherently not suitable for high precision real-time applications.

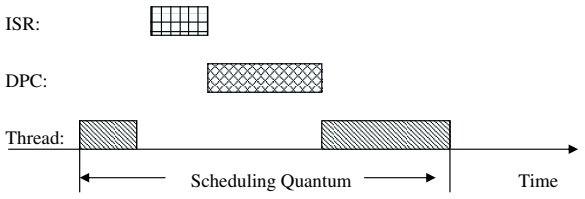

Figure 4: ISR/DPC Stealing CPU Cycles from Thread

3. REAL-TIME FOR NT AND XP EMBEDDED

The original requirement for real-time naturally came from embedded customers. Microsoft released Windows NT Embedded in 1999. NT Embedded quickly attracted many embedded customers because of its application compatibility with desktop Windows. Any application written for desktop windows can also run on NT/XP Embedded as long as supporting components for this application are included in the OS image. Embedded application development used to be notoriously hard due to lack of standards, tools and experienced developers, all of which are highly available for desktop Windows applications. NT Embedded quickly became a hit on the market.

Soon after the release of NT Embedded, customers started to ask for real-time support. Microsoft has hence partnered with several third-parties to provide real-time solutions. Examples include VentureCom (now Ardence)'s RTX [2], TenaSys' InTime [6], and Kuka Robotics's CeWin [3].

Regardless of who provides the new real-time kernel, they achieve real-time on Windows in a similar fashion. A new kernel (the real-time kernel) is introduced as a device driver in Windows or through a similar means. The new kernel exposes a different set of APIs from desktop Windows. Real time applications are written on this new API set and run with on priorities. Windows, on the other hand, always runs on the lowest priority. Namely, it runs when all real-time processes don't need to run. In a strict sense, none of these solutions makes Windows real-time. They just provide an additional OS to handle real-time applications side-by-side with Windows.

Notice that these real-time applications are not regular desktop Windows applications.

These solutions are used in scenarios with real-time data acquisition (as input) and non-time-critical data visualization (as output). Examples include medical devices, industrial control and military scenarios.

In comparison, there have been many initiatives to make Linux real-time. Two general categories of solutions have been proposed in academia and commercially implemented. The first one modifies the current Linux kernel or simply replaces it with a new one. The new kernel retains and

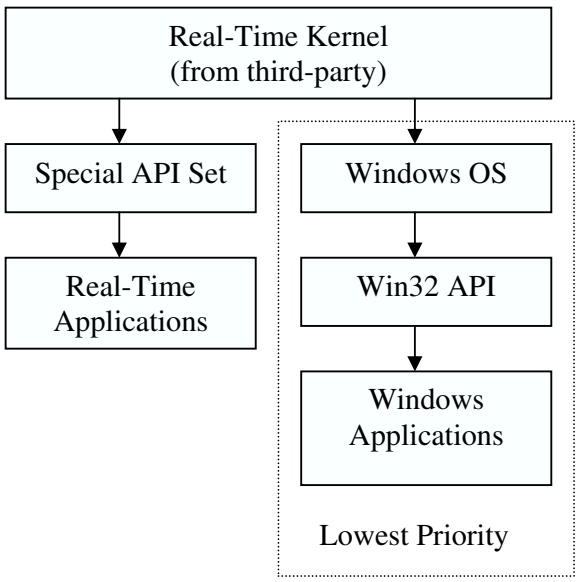

Figure 5: Real-Time Solution for Windows Embedded

implements the original kernel API with a full set of system calls. Examples using this approach include TimeSys (Linux/RK) [10] and Red-Linux [12]. The other approach imposes another level of kernel (called sub-kernel) on the top of the existing Linux kernel, similar to the Windows Embedded solutions. In this way, Linux is treated as the lowest priority task of the sub-kernel OS. RTLinux [5] and RTAI [4] are examples of this category.

4. WINDOWS VISTA

Windows Vista, formally codenamed "Longhorn" is the next version of Windows, which is scheduled for release at the second half of 2006. One of the great improvements in Windows Vista is on its AV quality, internally called "glitch free".

Media applications can be generally categorized as soft real-time. Desktop users often experience glitches, which include frame drop, frame duplicates, and AV Sync issues.

The goal of glitch free is two-fold. On one hand, we aim to eliminate glitches, including adding resilience to CPU, IO and memory stress. On the other hand, we also need to preserve the system stability, make sure that resources (CPU, memory etc) are used efficiently.

Towards these goals, several new kernel features are planned for Windows Vista:

- Multimedia Class Scheduler Service (MMCSS). MMCSS schedules Windows according to a machine-based policy. Different products (OEM SKUs) may differ in terms of the priorities that are assigned on different applications. For example, the policy allows individual OEMs to customize the task priorities on their machines to reflect the needs of target markets.

  To implement the policy, a thread scheduling service adjusts thread priorities to reflect the intentions of the policy. Threads associated with tasks listed in the policy as being of higher importance receive pri-

ority boosts which allow them to share limited CPU resources at relatively high priorities.

In the case of media application, the new scheduler assigns higher priorities to media processes than to other processes, while at the same time, ensure system processes' share to run.

- Thread-Ordering: The thread ordering service is responsible for controlling the execution timing of client threads. Each client thread belongs to a "thread ordering group". A thread ordering group has one and only one parent thread. The parent thread can have zero or more dependent threads. Each dependent thread may be a predecessor thread, meaning it must run before the parent thread, or a successor thread, meaning it must run after the parent thread. Each thread ordering group is run once per period, where the period is defined by the parent. The possible valid periods that the parent can specify are platform dependent but typically range from as low as 500 microseconds. At the start of each period, the service releases all predecessor threads. When all predecessor threads have finished their processing for that period, the parent thread is released. After the parent has completed its processing for that period, the service releases all successor threads.

- Real-Time Heap: Real-Time Heap is a new memory service providing non-blocking heap functions by allowing the locking of RT code pages in memory.

- Scheduled File I/O: addresses storage conflicts, allows bandwidth reservation, and allows prioritization. However, due to priority inversion and other resource contention issues, file IO still remains non-deterministic.

Although this new features have not been finalized on Vista, the efforts making Windows a more deterministic OS have been exemplified.

5. BEYOND WINDOWS VISTA

A huge paradigm shift of the evolution of processors beyond Windows Vista will be the multi-core processor technology. To keep up with the Moore's law, chip manufactures, mainly Intel and AMD, found it harder and harder to increase the processor frequency. Instead, they plan to continue increasing the number of cores on a processor socket. The movement from high frequency processors to multi-core processors is underway and will continue at least over the next decade.

Many factors contributed to the multi-core technology:

- Heat Dissipation. The higher the processor frequency gets, the more energy loss in the form of heat generation occurs. This leads to significant energy wasting both directly and indirectly as the cooling requirements increase. Meanwhile, new cooling systems such as liquid cooled ones are much more complicated than air cooled systems, thus costing much more .

- Disparity between processors and memory: With faster processors, the disparity between memory and processor speeds is larger so having a larger quantity of slower processors results in smaller Cycles Per Instruction (CPI) by reducing stall time.

- Parallelism: Multi-core processors with simpler instruction parallelism can lead to higher overall performance with considerably lower power consumption and heat generation.

- Form factor: With shrinking processor technologies, there is more room on a processor for multiple cores and processor caches.

Take the recently released AMD Dual Core Opteron processor for example [1].

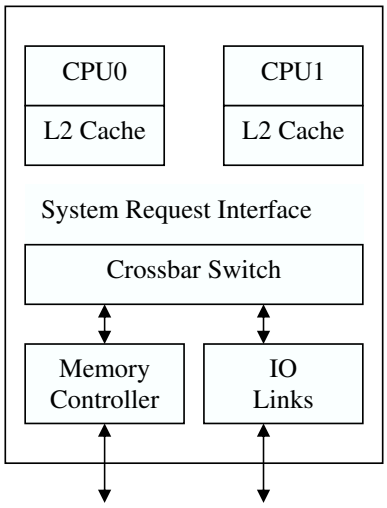

Figure 6: AMD Dual Core Opteron Architecture

The implications of multi-core technology are profound. Most obviously, it is extremely important to improve application scaling as soon as possible. This will also have a huge impact on real-time. With more and more cores on one processor, we will have higher predictability on a subset of cores. Random factors including ISRs, DPCs can be directed to the remaining cores. As long as we have exclusive access to a certain number of cores, providing guaranteed CPU allocation will be possible. True, we still need other parts of the system to work together, like memory and IO,.but multi-core will enable the key parts of a computer, namely the CPU to become real-time ready, and to lead to changes in other computer parts including memory and IO.

6. CONCLUSION

Real-Time is hard, hard real-time is even harder. However, they are not infeasible. With customer demands on one side and hardware evolution on the other, the horizon of real-time Windows is expanding. At the same time, a new set of challenging problems have thrown down the gauntlet to operating system researchers. How should applications be scheduled on multi-core processors to provide real-time guarantees? How should we effectively design caches for multi-core processors? How will multi-core lead the evolution of computer architecture? All these questions do not have definitive answers yet they will set the trend for generations of operating systems to come.

The author is grateful to all the people that provide resources, support and feedback to this paper, including Richard Russell, Michael Fortin, Brad Waters, Darryl Havens, Andy Glass, Stewart Tansley and Wayne Wolf.

7. REFERENCES

[1] Amd:http://www.amd.com/.

[2] Ardence:http://www.ardence.com/.

[3] Kuka robotics:http://www.kuka.com/.

[4] Rtai:http://www.rtai.org/.

[5] Rtlinux:http://www.fsmlabs.com/.

[6] Tenasys:http://www.tenasys.com/.

[7] Z. Deng and J. Liu. Scheduling real-time applications in an open environment. In *IEEE Real-Time Systems Symposium*, pages 308–319, December 1997.

[8] R. Mark. Mandatory voip 911 bills introduced. *http://www.internetnews.com/infra/article.php/3506741*, 2005.

[9] Microsoft. Real-time systems and microsoft windows nt: http://msdn.microsoft.com. *MSDN*, 1995.

[10] R. Rajkumar, L. Abeni, D. D. Niz, S. Gosh, A. Miyoshi, and S. Saewong. Recent developments with Linux/RK. In *Proceedings of the Real Time Linux Workshop*, December 2000.

[11] M. Russinovich and D. Solomon. *Windows Internals: Microsoft Windows Server 2003, Windows XP, and Windows 2000*. Microsoft Press, 2005.

[12] Y.-C. Wang and K.-J. Lin. Implementing a general real-time scheduling framework in the RED-linux real-time kernel. In *IEEE Real-Time Systems Symposium*, pages 246–255, 1999.