Results from a Latency Study of Windows NT

Or…

The Problems You're Having May Not Be the Problems You Think You're Having

Michael B. Jones
Microsoft Research, Microsoft Corporation
One Microsoft Way, Building 31/2260
Redmond, WA 98052, USA
mbj@microsoft.com
http://research.microsoft.com/~mbj/

John Regehr*
Department of Computer Science, Thornton Hall
University of Virginia
Charlottesville, VA 22903-2242, USA
regehr@virginia.edu
http://www.cs.virginia.edu/~jdr8d/

1. Introduction

This paper presents a snapshot of early results from a study of Windows NT aimed at understanding and improving its limitations when used for real-time tasks, such as those that arise for audio, video, and industrial control applications. It also examines the roles of intuition and conventional wisdom versus instrumentation and measurement in investigating latency behaviors.

Clearly there are time scales for which Windows NT can achieve effectively perfect reliability, such as the one-second deadlines present in the Tiger Video Fileserver [Bolosky et al. 97]. Other time scales, such as reliable sub-millisecond scheduling of periodic tasks in user space, are clearly out of reach. Yet, there is an interesting middle ground between these time scales in which deadlines may be met, but will not always be. Many useful real-time activities, such as fine-grained real-time audio waveform synthesis, fall into this middle range.

This study focuses on system and application behaviors in this region with the short-term goals of understanding and improving the real-time responsiveness of applications using Windows NT 5.0 and a longer-term goal of prototyping and recommending possible scheduling and resource management enhancements to future Microsoft systems products.

We present several examples of long scheduling latencies and the causes for them. While we went into the study with hunches about the causes of long latencies, these were almost always wrong. Only instrumentation and system measurement revealed the true causes.

2. Windows NT Background

Windows NT [Solomon 98] is increasingly being used to run time-dependent tasks. This is the case even though it was designed primarily to maximize aggregate throughput and to achieve approximately fair sharing of resources rather than to provide low-latency response to events, predictable scheduling, or explicit resource allocation mechanisms.

Features it does have include high-priority real-time thread priorities, interrupt routines that typically re-enable interrupts very quickly, and periodic callback routines.

Under Windows NT, not all CPU time is controlled by the scheduler. First, time spent handling interrupts is unscheduled. Also, much driver-related work occurs in *Deferred Procedure Calls* (DPCs) — routines executed within the kernel in no particular thread context in response to queued requests for their execution. Hardware interrupt latency is reduced by having interrupt handlers queue DPCs to finish the work associated with them. All queued DPCs are executed whenever a thread is selected for execution just prior to starting the selected thread. While good for interrupt latencies, DPCs can be bad for thread scheduling latencies, as they can potentially cause unbounded delays before a thread is scheduled.

Windows NT uses a loadable *Hardware Abstraction Layer* (HAL) module that isolates the kernel and drivers from low-level hardware details such as I/O interfaces, interrupt controllers, and multiprocessor communication mechanisms. The system clock is one service provided by each HAL. The HAL generates periodic clock interrupts for the kernel. The HAL interface contains no means of requesting a single interrupt at a particular time.

The Win32 interface contains a facility called *Multimedia Timers* supporting periodic execution of application code at a frequency specified by the application. The period is specified in 1ms increments. By default, the kernel receives a clock interrupt every 10 to 15ms; to permit more accurate timing, multimedia timers internally use a Windows NT system call that allows the timer interrupt frequency to be adjusted within the range permitted by the HAL (typically 1-15ms).

3. Baseline Performance Measurements

Given that Multimedia Timers are the primary mechanism available for applications to request timely execution of code, it is important for time-sensitive applications to understand how well it works in practice.

We wrote a test application that sets the clock frequency to the smallest period supported by the HAL (~1ms for all HALs used in these tests) and requests callbacks every 1ms. Each callback just records the Pentium cycle counter value at which it occurred in pinned memory. The application runs at the highest real-time priority. It is blocked waiting for a callback nearly 100% of the time, and so imposes no significant load on the system. The core of the application is as follows:

```
int main(…) {
    timeGetDevCaps(&TimeCap, …);
    timeBeginPeriod(TimeCap.wPeriodMin);
        // Set clock period to min supported
    TimerID = timeSetEvent(
        // Start periodic callback
        1,  // period (in milliseconds)
        0,  // resolution (0 = maximum)
        CallBack,   // callback function
        0,  // no user data
        TIME_PERIODIC);  // periodic timer
}
void Callback(…) {
    TimeStamp [i++] = ReadTimeStamp();
        // Record Pentium cycle counter value
}
```

On an ideal computer system dedicated to this program the callbacks would occur exactly 1ms apart. Actual runs allow us to determine how close real versions of Windows NT running on real hardware come to this.

Measurements were made on two different machines:

- a Pentium Pro 200MHz uniprocessor, with both an Intel EtherExpress 16 ISA Ethernet card and a DEC 21140 DC21x4-based PCI Fast Ethernet card, running uniprocessor kernels, using the standard uniprocessor PC HAL, HALX86.
- a Pentium 2 333MHz uniprocessor (but with a dual-processor motherboard) with an Intel EtherExpress Pro PCI Ethernet card, running multiprocessor kernels, using the standard multiprocessor PC HAL, HALMPS.

NT4 measurements were made under Windows NT 4.0, Service Pack 3. NT5 measurements were made under Windows NT 5.0, build 1805 (a developer build between Beta 1 and Beta 2). All measurements were made while attached to the network.

3.1 Supported Clock Rates

The standard uniprocessor HAL advertises support for clock rates in the range 1003µs to 14995µs. The actual rate observed during our tests was equal to the minimum, 1003µs. This was true for both NT4 and NT5.

The standard multiprocessor HAL advertises support for clock rates in the range 1000µs to 15625µs. The actual rate observed during our tests, however, was 976µs – less than the advertised minimum. See Section 4.1 for some of the implications of this fact. Once again, these observations were consistent across NT4 and NT5.

Finally, note that some HALs do not even support variable clock rates. This limits Multimedia Timer resolution to a constant clock rate chosen by the HAL.

3.2 Times Between Timer Callbacks

Table 1 gives statistics for typical 10-second runs of the test application on both test machines for both operating system versions.

Times Between Callbacks	**PPro, NT4**	**PPro, NT5**	**P2, NT4**	**P2, NT5**
Minimum µs	31	31	20	33
Maximum µs	2384	18114	2144	2396
Average µs	999	999	999	999
Std Dev µs	70	211	955	941

Table 1: Statistics about Times Between Callbacks

All provide an average time between callbacks of 999µs, but the similarities end there. Note, for instance, that the standard deviation for the Pentium 2 runs is around 950µs – nearly equal to the mean! Also, notice that there was at least one instance on the Pentium Pro under NT5 when no callback occurred for over 18ms.

The statistics do not come close to telling the full story. Table 2 is a histogram of the actual times between callbacks for these same runs, quantized into 100µs bins.

# Times Between Callbacks Falling Within Interval	**PPro, NT4**	**PPro, NT5**	**P2, NT4**	**P2, NT5**
0-100µs	34	62	4880	4880
100-200µs	1			
300-400µs		1		
500-600µs	4	2		
600-700µs	6	1		
700-800µs	22			
800-900µs	150	10		
900-1000µs	571	1281		
1000-1100µs	9014	8627		
1100-1200µs	161	10		
1200-1300µs	28	1		
1300-1400µs	6	1		
1400-1500µs	1	1		2
1700-1800µs			2	5
1800-1900µs			9	91
1900-2000µs			5107	5014
2000-2100µs				4
2100-2200µs			2	2
2300-2400µs	2			2
7700-7800µs		2		
18100-18200µs		1		

Table 2: Histogram of Times Between Callbacks

Now, the reason for the high standard deviation for the Pentium 2 runs is clear – no callbacks occurred with spacings anywhere close to the desired 1ms apart. Instead, about half occurred close to 0ms apart and half occurred about 2ms apart!

Also, for the Pentium Pro NT5 run, note that twice callbacks occurred about 7.7ms apart and once over 18ms apart. In fact, this is not atypical. On this configuration, there are *always* two samples around 7-8ms apart and one around 18ms apart.

Indeed, the point of our study is to try to learn what is causing anomalies such as these, and to fix them!

4. Problems and Non-Problems

4.1 Problem: HAL Timing Differences

Because the HAL virtualizes the hardware timer interface, HAL writers may implement timers in different ways. For example, HALX86 uses the 8254 clock chip to generate clock interrupts on IRQ1, but HALMPS uses the Real Time Clock (RTC) to generate interrupts on IRQ8.

Upon receiving a clock interrupt, the HAL calls up to the Windows NT kernel, which (among other things) compares the current time to the expiration time of any pending timers, and dequeues and processes those timers whose expiration times have passed.

As we have seen, multimedia timers are able to meet 1ms deadlines most of the time on machines running HALX86. To understand why 1ms timers do not work on machines running HALMPS, we next examine the timer implementation in more detail.

A periodic multimedia timer always knows the time at which it should next fire; every time it does fire, it increments this value by the timer interval. If the next firing time is ever in the past, the timer repeatedly fires until the time has moved into the future. The next firing time is rounded to the nearest millisecond. This interacts poorly with HALMPS, which approximates 1ms clock interrupts by firing at 1024Hz, or every 976µs. (The RTC only supports power-of-2 frequencies.)

Because the interrupt frequency is slightly higher than the timer frequency, we would expect to occasionally wait almost 2ms for a callback when the 976µs interrupt interval happens to be contained within the 1000µs timer interval. Unfortunately, rounding the firing time ensures that this worst case becomes the common case. Since it never asks to wait less than 1ms, it always waits nearly 2ms before expiring, then fires again immediately to catch up, hence the observed behavior.

We fixed this error by modifying the timer implementation to compute the next firing time more precisely, allowing it to request wakeups less than 1ms in the future. (An alternative fix would have been to use periodic kernel timers, rather than repeatedly setting one-shot timers.) The results of our fix can be seen in Table 3.

As expected, approximately 2.4% of the wakeups occur near 2ms, since clock interrupts arrive 2.4% faster than timers. As a number of HALs besides HALMPS use the RTC, this fix should be generally useful.

# Times Between Callbacks Falling Within Interval	**P2, NT5**	**P2, NT5 fixed**
0-100µs	4880	1
500-600µs		1
600-700µs		3
700-800µs		2
800-900µs		7
900-1000µs		9609
1000-1100µs		127
1100-1200µs		4
1200-1300µs		2
1300-1400µs		2
1400-1500µs	2	2
1700-1800µs	5	
1800-1900µs	91	
1900-2000µs	5014	240
2000-2100µs	4	
2100-2200µs	2	
2300-2400µs	2	

Table 3: Histogram Showing Results of Timer Fix

4.2 Non-Problem: Interrupts

One piece of conventional wisdom is that the problems might be caused by interrupts. Yet we never observed an interrupt handler taking substantial fraction of a millisecond. We believe this is the case since interrupts needing substantial work typically queue DPCs to do their work in a non-interrupt context.

4.3 Non-Problem: Ethernet Receive Processing

Another commonly held view is that Ethernet input packet processing is a problem. Yet we tested many of the most popular 10/100 Ethernet cards receiving full rate 100Mbit point-to-point TCP traffic up to user state. The cards we tested were the Intel EtherExpress Pro 100b, the SMC EtherPower II 10/100, the Compaq Netelligent 10/100 Tx, and the DEC dc21x4 Fast 10/100 Ethernet. The longest observed individual DPC execution we observed was only 600 µs, and the longest cumulative delay of user-space threads was approximately 2ms. Ethernet receive processing may have been a problem for dumb ISA cards on 386/20s, but it's no longer a problem for modern cards and machines.

4.4 Problem: Long-Running DPCs

However, we did find numerous network-related latency problems caused by "unimportant" background work done by the cards or their drivers in DPCs.

DEC dc21x4 PCI Ethernet Card

Through instrumentation, we were able to determine that the 7.7ms delays on the Pentium Pro were caused by a long-running DPC. In particular, the DEC dc21x4 PCI Fast 10/100 Ethernet driver causes a periodic DPC to be executed every 5 seconds to do autosense processing (determining if the card is connected to a 10Mbit or 100Mbit Ethernet). And this "unimportant background work" takes 6-7 ms every five seconds.

This is largely due to poor hardware design. In particular, most of this delay is occurs when the driver does bit-serial reads and writes to three 16-bit status registers, with 5µs stalls per bit, 48 in all.

Intel EtherExpress 16 ISA Ethernet Card

Similarly, the Intel EtherExpress 16 (EE16) ISA Ethernet card and driver caused the 18.1ms delay. Every ten seconds it schedules a DPC to wake up and reset the card if no packets have been received during the past ten seconds. Why? Because some versions of the card would occasionally lock up and resetting them would make them usable again. Probably no one thought that the hardware reset path had to be fast. And it isn't! It takes 17ms.

An amusing observation about this scenario is that the conventional wisdom is that unplugging your Ethernet will make your machine run more predictably. But for this driver, unplugging your Ethernet makes latency worse! Once again, your intuition will lead you astray.

4.5 Problem: Antisocial Video Cards

Misbehaving video card drivers are another source of significant delays in scheduling user code. A number of video cards manufacturers recently began using a hack to save a PCI bus transaction for each display operation in order to gain a few percentage points on their WinBench [Ziff-Davis 98] Graphics WinMark performance.

The video cards have a command FIFO that is written to via the PCI bus. They also have a status register, read via the PCI bus, which says whether the command FIFO is full or not. The hack is to not check whether the command FIFO is full before attempting to write to it, thus saving a PCI bus read.

The problem with this is that the result of attempting to write to the FIFO when it is full is to stall the CPU waiting on the PCI bus write until a command has been completed and space becomes available to accept the new command. In fact, this not only causes the CPU to stall waiting on the PCI bus, but since the PCI controller chip also controls the ISA bus and mediates interrupts, ISA traffic and interrupt requests are stalled as well. Even the clock interrupts stop.

These video cards will stall the machine, for instance, when the user drags a window. For windows occupying most of a 1024x768 screen on a 333MHz Pentium II with an AccelStar II AGP video board (which is based on the 3D Labs Permedia 2 chip set) this will stall the machine for 25-30ms at a time!

This may marginally improve the graphics performance under some circumstances, but it wrecks havoc on any other devices expecting timely response from the machine. For instance, this causes severe problems with USB and IEEE 1394 video and audio streams, as well as standard sound cards.

Some manufacturers, such as 3D Labs, do provide a registry key that can be set to disable this anti-social behavior. For instance, [Hanssen 98] describes this behavior and lists the registry keys to fix several common graphics cards, including some by Matrox, Tseng Labs, Hercules, and S3. However as of this writing, there were still drivers, including some from Number 9 and ATI, for which this behavior could not be disabled.

This hack, and the problems it causes, has recently started to receive attention in the trade press [PC Magazine 98]. We hope that pressures can soon be brought to bear on the vendors to cease this antisocial behavior. At the very least, should they persist in writing drivers that can stall the machine, this behavior should no longer be the default.

5. Conclusions

While the essential structure of Windows NT is capable of providing low-latency response to events, obvious (and often easy to fix!) problems we have seen, such as video drivers that intentionally stall the PCI bus, the poor interaction between multimedia timers and HALMPS and occasional long DPC execution times, keep current versions of Windows NT from guaranteeing timely response to real-time events below thresholds in the tens of milliseconds. Bottom line – the system is clearly not being actively developed or tested for real-time responsiveness. We are working to change that!

Finally, our experiences during this study only reinforce the truth that instrumentation and measurement is the only way to truly understand the performance of computer systems. Intuition will lead you astray.

References

[Bolosky et al. 97] William J. Bolosky, Robert P. Fitzgerald, and John R. Douceur. Distributed Schedule Management in the Tiger Video Fileserver. In *Proc. of the 16th ACM Symposium on Operating Systems Principles*, St-Malo, France, pp. 212-223, Oct. 1997.

[Hanssen 98] Greg Hanssen. *vgakills.txt*. http://www.zefiro.com/vgakills.txt, Zefiro Acoustics, February, 1998.

[PC Magazine 98] PC Magazine Online. *Inside PC Labs: PCI Problems*. http://www.zdnet.com/pcmag/news/trends/t980619a.htm, Ziff-Davis, June 19, 1998.

[Solomon 98] David A. Solomon. *Inside Windows NT, Second Edition.* Microsoft Press, 1998.

[Ziff-Davis 98] *WinBench 98*. http://www.zdnet.com/zdbop/winbench/winbench.html, Ziff-Davis, 1998.