Windows Timer Coalescing

January 20, 2009

Abstract

This paper provides information about timer coalescing for the Windows® family of operating systems. Timer coalescing helps improve the energy efficiency of periodic software activity by expiring multiple distinct software timers at the same time to increase the average processor idle period. This paper provides guidelines for driver and application developers to take advantage of the new Windows functions for timer coalescing.

This information applies for the following operating systems:  
 Windows Server® 2008 R2  
 Windows 7

References and resources discussed here are listed at the end of this paper.

For the latest information, see:  
 <http://www.microsoft.com/whdc/system/pnppwr/powermgmt/TimerCoal.mspx>

Disclaimer: This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2009 Microsoft Corporation. All rights reserved.

Microsoft, Windows, and Windows Server are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Document History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Change |  |  |  |
| January 20, 2009 | First publication | | | |

Contents

[Introduction 3](#_Toc220139802)

[Reducing Power Consumption with Timer Coalescing 3](#_Toc220139803)

[Processor Power Consumption and Idle Periods 3](#_Toc220139804)

[Timer Coalescing Fundamentals 6](#_Toc220139805)

[Kernel-Mode Timer Coalescing Function 6](#_Toc220139806)

[Timer Coalescing within a WDF Driver 7](#_Toc220139807)

[User-Mode Timer Coalescing Function 8](#_Toc220139808)

[Best Practices for Using Timer Coalescing 8](#_Toc220139809)

[Resources 9](#_Toc220139810)

# Introduction

System power consumption and energy efficiency are heavily influenced by the amount of processor activity, including periodic activity from applications and device drivers. Modern processors can reduce their power consumption by entering into a low‑power idle state during the periods of idle time between executing instructions for software activity. However, many processor power management technologies require a minimum amount of idle time to obtain a net power-savings benefit. If the processor is idle for only very short periods of time, the power that is required to enter and exit the low-power state can be greater than the power that is saved.

Periodic software activity and software timers heavily influence the amount of processor idle time and the duration of the idle periods. Windows® timer coalescing helps increase the duration of idle periods by combining different software timer expirations so that they all occur on a single timer interrupt.

We recommend that software developers evaluate their code first for opportunities to remove periodic activity. If it is possible, periodic activity should be changed to event-driven or interrupt-based designs. However, if periodic activity is required, the code should use timer coalescing to help enable the Windows kernel to more efficiently expire the associated timer.

This paper describes Windows timer coalescing and how innovations in the Windows kernel for managing the platform timer interrupt help increase processor idle periods and therefore improve overall system energy efficiency. Additionally, kernel-mode timer and user-mode timer coalescing functions are described, together with best practices for taking advantage of timer coalescing in software.

# Reducing Power Consumption with Timer Coalescing

On portable computers, reducing processor usage and extending idle periods are key to improving platform energy efficiency and extending battery life. Combined with other Windows 7 kernel innovations, timer coalescing can help increase the average idle period of system processors. Timer coalescing lets the Windows kernel combine periodic software timer activity to increase the average idle period.

You should use timer coalescing for periodic software activity that cannot be changed to an interrupt or event-driven design. Software timers that use timer coalescing should specify a minimum of 32 milliseconds (ms) for the timer expiration tolerance.

## Processor Power Consumption and Idle Periods

The amount of processor activity directly correlates to processor and platform power consumption. This is obvious when you observe the platform power consumption as you play a 3‑D game and compare it to the power consumption as you type a document in a word processing application. Typically, you should first optimize platforms for their idle power consumption, which is the amount of power that is consumed when no user-interactive activity is occurring. Idle power consumption serves as the basis for power consumption across all other scenarios, such as DVD playback, productivity, media creation, or gaming. Therefore, reducing the idle power consumption improves the power consumption of all other system scenarios. Best-in-class systems have total idle processor usage of less than 2 percent, as measured by the Windows Performance Analysis Toolkit (XPerf).

However, total processor usage is not the only important metric for reducing idle power consumption. The duration of idle time between software instructions or platform break events is also important. Modern processors and chipsets achieve extremely low power consumption by transitioning into a low‑power idle state during periods of idle time. However, they can enter low‑power idle states only if the duration of the idle time is sufficient compared to the power and latency of entering and exiting the low-power idle state.

Consider the example of processor activity that is shown in Figure 1. This example has two primary sources of processor activity: the platform timer interrupt that drives the Windows kernel scheduler and intermittent software timer activity. The intermittent software timer activity is the result of the Windows kernel expiring the software timers after each timer interrupt.



Figure 1. Platform timer interrupt and software timers

On a system that has multiple logical processors, the behavior is mirrored across each processor in the system. Each processor must receive the updated platform timer value and expire any software timers that have come due. Figure 2 shows an example of platform timer interrupts and software timer expiration on a multiprocessor system.



Figure 2. Multiprocessor timer expiration

In Windows 7 and later versions of Windows, the kernel is improved so that the platform timer interrupt does not automatically wake application processors. Application processors (APs) are any other processors in the system that are not the primary or base service processor (BSP). This improvement prevents APs from waking from low-power idle states unless software timers are expiring or hardware interrupts occur other than the platform timer interrupt. Letting the APs remain in the idle power state for longer durations of time helps reduce processor and platform power consumption. Figure 3 shows an example where the platform timer interrupts are removed from the APs as a result of these kernel improvements.

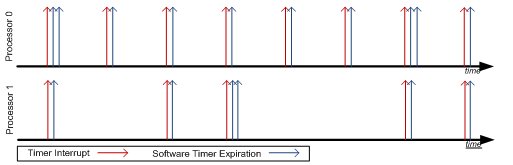


Figure 3. Windows 7 improvements in timer interrupt delivery to APs

Timer coalescing is the ability of the Windows kernel to group multiple software timers from different applications and device drivers and expire them all at the same time. When you combine timer coalescing with the platform timer interrupt distribution improvements in Windows 7, timer coalescing further helps increase the idle duration of the APs. This helps save processor and platform power by waking each AP only when it needs to perform the work for multiple software timers. After that work is complete, the processor returns to a low-power idle state for an extended period of time.

Figure 4 shows the advantages of timer coalescing when it is combined with removing platform timer interrupts from APs.



Figure 4. Timer expiration with timer coalescing

In this example, processor 1 has very long periods of idle time and processor 0 has coalesced software timer expirations to fewer platform timer interrupts.

## Timer Coalescing Fundamentals

Timer coalescing enables application and device driver software to specify a *tolerable delay* for the expiration of a software timer. The Windows kernel uses the tolerable delay value to adjust the time that the timer is due to expire so that it coincides with the expiration time of other software timers.

Although timer coalescing does help improve the average processor idle duration and therefore reduce the overall system power consumption, you should first eliminate any unnecessary periodic activity in your software. You should use timer coalescing for software timers only if you cannot change the periodic activity to an interrupt or event-driven design.

Generally, software should specify a minimum tolerable delay of 32 ms, which corresponds to the duration of two default system clock intervals of 15.6 ms each. If it is possible, a larger tolerable delay, such as 50 ms, is preferred. Also, the tolerable delay can often scale up with the period of the timer. For example, a 500‑ms timer might use a 50-ms tolerable delay, whereas a 10-second timer could use a 1‑second tolerable delay.

To help improve system efficiency when expiring software timers that can be coalesced, we encourage you to specify both the period and the tolerable delay for the timers in multiples of 50 ms. For example, you should use timer periods of 50, 100, 250, 500, and 1,000 ms. Similarly, tolerable delay values of 50, 100, 150, and 250 ms are appropriate.

The key consideration when you migrate periodic activity to use software timers that can be coalesced is that the period between successive timer expirations is not guaranteed. The duration of time between any two timer expirations is within the range of the period of the timer, plus or minus the tolerable delay.

# Kernel-Mode Timer Coalescing Function

You can take advantage of Windows timer coalescing in device drivers by using the new kernel-mode **KeSetCoalescableTimer** function. However, we recommend that you first determine whether you can remove any periodic activity from your drivers and change to interrupt or event-driven designs. If you cannot eliminate the periodic activity, Windows timer coalescing helps the periodic activity become more efficient.

The following is the function prototype for the **KeSetCoalescableTimer**function:

NTKERNELAPI

BOOLEAN

KeSetCoalescableTimer (

\_\_inout PKTIMER Timer,

\_\_in LARGE\_INTEGER DueTime,

\_\_in ULONG Period,

\_\_in ULONG **TolerableDelay**,

\_\_in\_opt PKDPC Dpc

);

**KeSetCoalesableTimer**resembles the **KeSetTimerEx** function that you use to set the period when a timer should periodically expire. In many situations you can easily replace existing calls to **KeSetTimerEx** with calls to **KeSetCoalescableTimer***.* The *TolerableDelay* parameter is the only new parameter for the **KeSetCoalescableTimer**function. This parameter lets the caller specify the tolerance for the timer's period in milliseconds.

You should specify a tolerance of at least 32 ms for the *TolerableDelay* parameter. Optimally, the tolerance scales up with the period of the timer. For example, if a timer has a period of 1 second, a tolerable delay of at least 50 ms is appropriate. However, if a timer has a period of 30 seconds, it should have a tolerable delay of at least 1,000 ms.

The following table describes the requirements for calling the **KeSetCoalescableTimer** function.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| KeSetCoalescableTimer requirements | |  |  |  |
| Interrupt request level (IRQL) | <= DISPATCH\_LEVEL | | | |
| Headers | Declared in *wdm.h*. Include *wdm.h*, *ntddk.h*, or *ntifs.h*. | | | |
| Supported operating systems | Windows 7 or Windows Server® 2008 R2 and later versions. | | | |

You can also use an *IoTimer* routine, which is a deferred procedure call (DPC) that is called one time per second. The underlying timer for an *IoTimer* routine is automatically coalesced in Windows 7 and later versions of Windows with a minimal (32‑ms) tolerable delay.

# Timer Coalescing within a WDF Driver

Windows timer coalescing is available in the Windows Driver Framework (WDF) beginning with version 1.9. You can specify a tolerable delay value in the **TolerableDelay** member of the WDF\_TIMER\_CONFIG structure. WDF version 1.9 defines the WDF\_TIMER\_CONFIG structure as follows:

typedef struct \_WDF\_TIMER\_CONFIG {  
  ULONG  Size;  
  PFN\_WDF\_TIMER  EvtTimerFunc;  
  ULONG  Period;  
  BOOLEAN  AutomaticSerialization;  
  ULONG  **TolerableDelay**;  
} WDF\_TIMER\_CONFIG, \*PWDF\_TIMER\_CONFIG;

The following table describes the requirements for using the **TolerableDelay** member of the WDF\_TIMER\_CONFIG structure.

|  |  |  |
| --- | --- | --- |
| WDF\_TIMER\_CONFIG.TolerableDelay requirements | |  |
| WDF version | WDF version 1.9 and later. | |
| Headers | Declared in *wdftimer.h*. Include *wdf.h*. | |
| Supported operating systems | Windows 7 or Windows Server 2008 R2 and later versions. | |

# User-Mode Timer Coalescing Function

You can take advantage of Windows timer coalescing in applications and services by using the new user-mode **SetWaitableTimerEx** function. However, similar to device drivers, we recommend that you first determine whether you can remove any periodic activity from your applications and services and change to event-driven designs. If you cannot eliminate the periodic activity, Windows timer coalescing helps the periodic activity become more efficient.

The following is the function prototype for the **SetWaitableTimerEx** function:

BOOL

WINAPI

SetWaitableTimerEx(

\_\_in HANDLE hTimer,

\_\_in const LARGE\_INTEGER \*lpDueTime,

\_\_in LONG lPeriod,

\_\_in\_opt PTIMERAPCROUTINE pfnCompletionRoutine,

\_\_in\_opt LPVOID lpArgToCompletionRoutine,

\_\_in\_opt PREASON\_CONTEXT WakeContext,

\_\_in ULONG **TolerableDelay**

);

**SetWaitableTimerEx** resembles the **SetWaitableTimer** function that you use to set the period when a timer should periodically expire. In many situations you can easily replace existing calls to **SetWaitableTimer** with calls to **SetWaitableTimerEx**. **SetWaitableTimerEx** has two new parameters: *WakeContext* and *TolerableDelay*. You use the *WakeContext* parameter only when you set a timer that can wake the system from a sleep state. The *TolerableDelay* parameter lets the caller specify the tolerance for the timer's period in milliseconds.

You should specify a tolerance of at least 32 ms for the *TolerableDelay* parameter. Optimally, the tolerance scales up with the period of the timer. For example, if a timer has a period of 1 second, a tolerable delay of at least 50 ms is appropriate. However, if a timer has a period of 30 seconds, it should have a tolerable delay of at least 1,000 ms.

The following table describes the requirements for calling the **SetWaitableTimerEx** function.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SetWaitableTimerEx requirements | |  |  |  |
| Header | Winbase.h | | | |
| Library | Kernel32.lib | | | |
| DLL | Kernel32.dll | | | |
| Supported operating systems | Windows 7 or Windows Server 2008 R2 or later versions. | | | |

# Best Practices for Using Timer Coalescing

You should use Windows timer coalescing to help improve the energy efficiency of your application and device driver software. However, timer coalescing cannot replace due diligence. You should first eliminate any unnecessary periodic activity in your software and change that activity to event-driven or interrupt-based designs.

When you use the Windows timer coalescing functions, you should use the following best practices:

* Review application and driver software for opportunities to take advantage of timer coalescing. Remember that the period for a coalesced timer is not guaranteed. However, successive timer expirations are always within plus or minus the tolerable delay of the specified period.
* Specify a value for the *TolerableDelay* parameter of at least 32 ms, which corresponds to two 15.6‑ms platform timer interrupt intervals.
* Scale up the tolerable delay with the period of the timer. For example, if a tolerable delay of 50 ms is appropriate for a timer that has a 500‑ms period, then a tolerable delay of 100 ms is appropriate for a timer that has a 1‑second period.
* Specify tolerable delay values and timer periods in multiples of 50 ms, for example, 50, 100, 250, 500 ms, and so on.
* If possible, change periodic activity in device drivers to use an *IoTimer* routine, which is called one time per second and is automatically coalesced by the Windows kernel.

# Resources

IoTimer

<http://msdn.microsoft.com/en-us/library/ms795711.aspx>