Timers, Timer Resolution, and Development of Efficient Code

June 16, 2010

Abstract

This paper provides information about high-resolution timers and periodic timers for Windows® operating systems. It provides guidelines for developers to use timers efficiently with platform power management. It assumes that the reader is familiar with concepts of periodic activity and scheduled timers.

This information applies to the following operating systems:

Windows Server® 2008 R2
Windows 7

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

The current version of this paper is maintained on the Web at:
http://www.microsoft.com/whdc/system/pnppwr/powermgmt/Timer-Resolution.mspx

Document History

Date	Change
June 16, 2010	First publication

Contents

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.

Software developers should evaluate their code for opportunities to remove periodic activity. If possible, periodic activity should be replaced with event-driven or interrupt-based designs. However, if periodic activity is required, this paper provides a summary of developer best practices that enable the Windows kernel to more efficiently manage timers and timer resolution.

Timer Resolution

The system timer resolution determines how frequently Windows performs two main actions:

- Update the timer tick count if a full tick has elapsed.
- Check whether a scheduled timer object has expired.

A timer tick is a notion of elapsed time that Windows uses to track the time of day and thread quantum times. By default, the clock interrupt and timer tick are the same, but Windows or an application can change the clock interrupt period.

The default timer resolution on Windows 7 is 15.6 milliseconds (ms). Some applications reduce this to 1 ms, which reduces the battery run time on mobile systems by as much as 25 percent.

Such a dramatic reduction in battery life is not desirable, yet the effect of changing timer resolution on desktop or server systems can be equally problematic. Decreased battery life on a mobile system makes increased power consumption visible. A system that is running on AC power incurs the same increased power usage, which may not be as simple to detect. The cost increase to a facility that has several thousand systems is significant.

Timer Coalescing

In addition to enabling systems to effectively use idle power states, improvements that were introduced in Windows 7 allow developers to take greater advantage of this technology. One of the tasks that the operating system undertakes on a system timer event is to check whether scheduled timers have expired. If so, a callback is made to the function pointer that is associated with the timer object.

Windows 7 introduced timer coalescing, which allows scheduled timers to specify a tolerable delay for timer expiration. For example, a scheduled timer that has a period of 200 ms may specify a tolerance of 20 ms. This allows Windows to group multiple software timer expirations into a single period of processing.

Figures 1, 2, and 3 illustrate this feature. Figure 1 shows a time line of clock interrupts and timers expiring. In Figure 2, the timer expirations are placed at the point where the system would issue a callback. The advantages of timer coalescing are shown in Figure 3, in which the timer expirations are grouped as might be possible with coalescing.

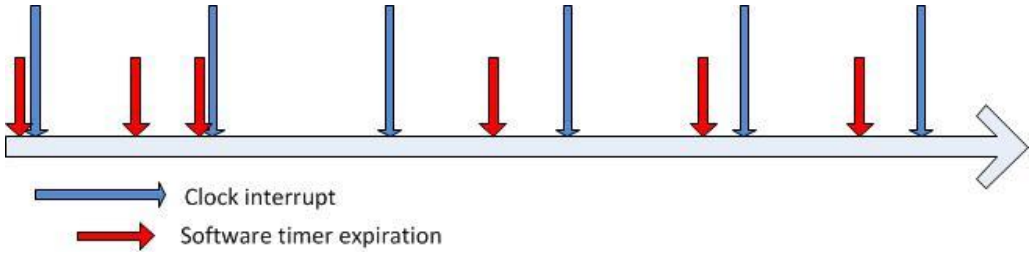

Figure 1. Expiration time of software timers

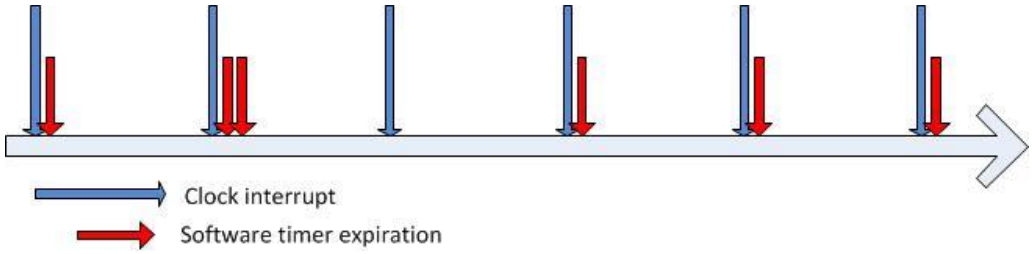

Figure 2. Service of expired timers with no coalescing

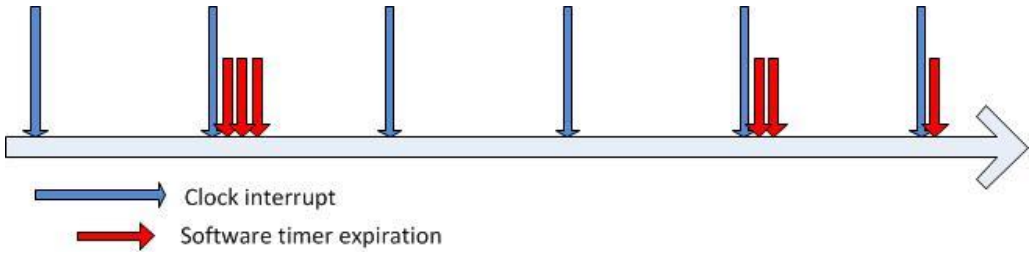

Figure 3. Service of expired timers with coalescing

The first scheduled timer has a tolerance that allows it to be delayed until the next timer interrupt, which lets the processor spend more time at idle after the first clock interrupt. Similarly, the fourth scheduled timer expiration is delayed until the next clock interrupt, which allows two timer expirations to be grouped.

Grouping processing of timer expiration in this way increases the number of periods between clock interrupts that have long idle time. This lets the system take full advantage of its power management features during these idle periods. As further hardware advances that reduce power are introduced, the advantages of having the processor in an idle state for extended periods will increase. Whenever it is possible,

developers should use timer coalescing so that applications are prepared for the future.

Intelligent Timer Tick Distribution (ITTD) is an example of an advance that increases energy savings when combined with timer coalescing. Introduced in Windows 7, ITTD is a kernel improvement that reduces clock interrupts on systems that have multiple logical processors. The change that ITTD introduced is that the platform timer interrupt does not automatically wake application processors.

Application processors (APs) are any processors in the system that are not the primary or base service processor (BSP). ITTD 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 increased durations of time helps reduce processor and platform power consumption.

As figures 4 and 5 illustrate, in addition to longer idle periods for the BSP, timer coalesce allows the platform timer interrupts to be removed from the AP.

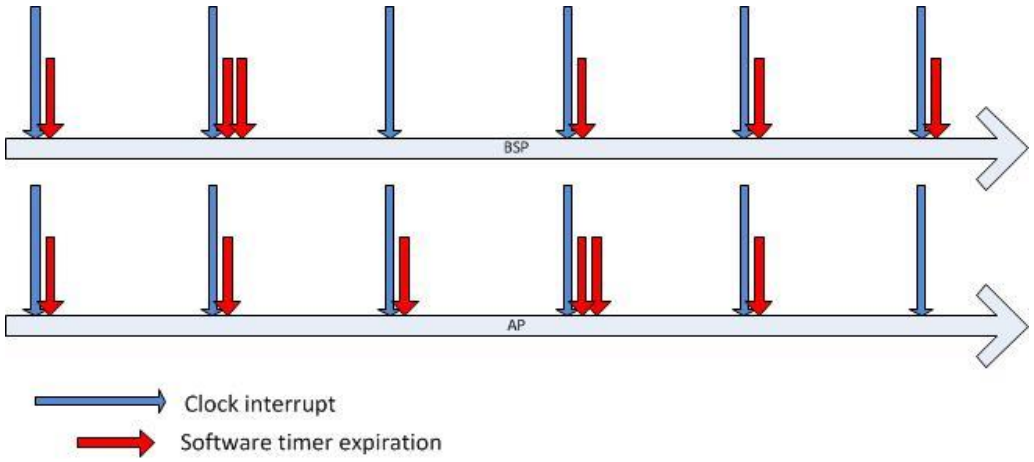

Figure 4. Multiple logical processors without coalesce

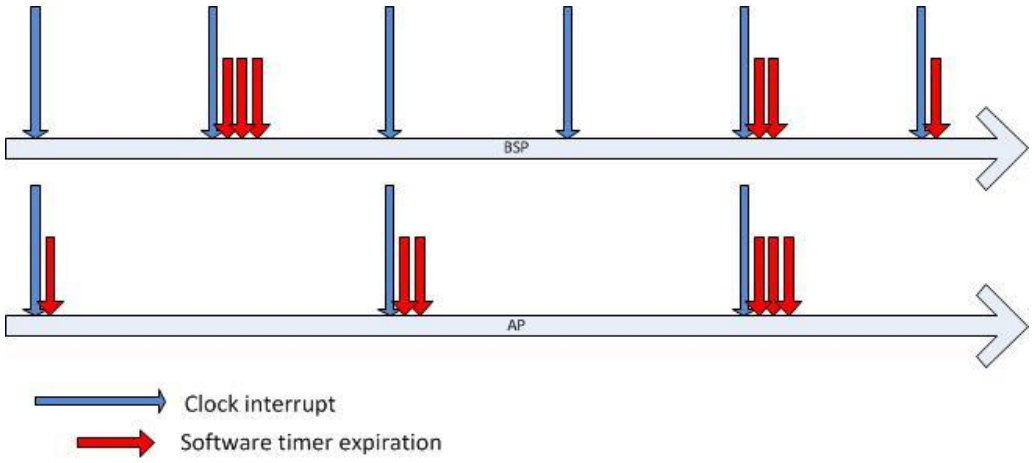

Figure 5. Multiple logical processors with coalescing and ITTD

By following the best practices for timer resolution, developers can avoid unnecessary battery drain and energy consumption, which reduces costs and improves mobile productivity. Using timer coalescing provides further enhancements to power management.

System Timer Resolution Manipulation

The use of high-resolution periodic timers in an application can increase overall system power consumption and subsequently reduce system battery life. Developers might choose to use high-resolution periodic timers through the Windows Multimedia API, or an application might use these timers inadvertently through external libraries and application development frameworks.

The default system-wide timer resolution in Windows is 15.6 ms, which means that every 15.6 ms the operating system receives a clock interrupt from the system timer hardware. When the clock interrupt fires, Windows updates the timer tick count and checks whether a scheduled timer object has expired. Some applications require that timer expiration is serviced more frequently than once every 15.6 ms.

Applications can call timeBeginPeriod to increase the timer resolution. The maximum resolution of 1 ms is used to support graphical animations, audio playback, or video playback. This not only increases the timer resolution for the application to 1 ms, but also affects the global system timer resolution, because Windows uses at least the highest resolution (that is, the lowest interval) that any application requests. Therefore, if only one application requests a timer resolution of 1 ms, the system timer sets the interval (also called the “system timer tick”) to at least 1 ms. For more information, see “**timeBeginPeriod** Function” on the MSDN® website.

Modern processors and chipsets, particularly in portable platforms, use the idle time between system timer intervals to reduce system power consumption. Various processor and chipset components are placed into low-power idle states between timer intervals. However, these low-power idle states are often ineffective at lowering system power consumption when the system timer interval is less than the default.

If the system timer interval is decreased to less than the default, including when an application calls **timeBeginPeriod** with a resolution of 1 ms, the low-power idle states are ineffective at reducing system power consumption and system battery life suffers.

System battery life can be reduced as much as 25 percent, depending on the hardware platform. This is because transitions to and from low-power states incur an energy cost. Therefore, entering and exiting low-power states without spending a minimum amount of time in the low-power states can be more costly than if the system simply remained in the high-power state.

If an application changes the timer resolution, developers should ensure that external libraries and application development frameworks do not unexpectedly also change the system timer interval on the application’s behalf.

Developers can determine whether an application increases the platform timer resolution by using the PowerCfg command-line utility with the **/energy** option. Run

the PowerCfg utility while the application to be analyzed is active, and then check the resulting energy report. The report lists the instances of increased platform timer resolution and indicates whether processes that are related to an application increased the timer resolution. Although a code scan reveals whether an application uses the **timeBeginPeriod** method, calls to external libraries or plug-ins may also alter timer resolution. Exercising a broad set of application scenarios while PowerCfg is monitoring timer resolution can be useful to locate timer issues.

Recommendations for Application Developers

When designing and developing applications, follow these best practices when using high-resolution periodic timers:

- ✓ Understand the effect of high-resolution periodic timers on system power consumption.

  A single application can cause the system timer interval to change, which can reduce system battery life as much as 25 percent.

- ✓ Validate that the system timer interval does not unexpectedly change the application that is running.

  To do this, run the PowerCfg utility with the **/energy** option and view the resulting energy report for instances of timer resolution increase requests.

  Specifying **/duration** 5 reduces the scan time to 5 seconds from the full 30-second default. Some systems have permissions issues with writing the energy report to the %systemroot%\system32 directory. Use the **/output** switch to redirect to a different location\file name.

- ✓ If an application must use a high-resolution periodic timer, enable the periodic timer only while the required functionality is active.

  For example, if the high-resolution periodic timer is required for animation, disable the periodic timer when the animation is complete. If the high-resolution periodic timer is required for audio or video playback, consider disabling the timer when:

  - Playback is paused.
  - The window or tab that contains the video player is not visible.
  - The screen is dimmed or sleeping.

- ✓ If an application must use a high-resolution periodic timer, consider disabling use of the periodic timer and associated functionality when a Power Saver power plan is active or when the system is running on battery power.

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 expired so that it coincides with the expiration time of other software timers.

Although timer coalescing helps improve the average processor idle duration and therefore reduce the overall system power consumption, developers should first eliminate any unnecessary periodic activity in their software. Use timer coalescing for

software timers only if it is impossible to change the periodic activity to an interrupt or event-driven design.

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 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 improve system efficiency for software timers that can be coalesced, developers can specify both the period and the tolerable delay for the timers in multiples of 50 ms. For example, 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 developers 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

Developers can take advantage of Windows timer coalescing in device drivers by using the new kernel-mode **KeSetCoalescableTimer** function. However, developers should first determine whether they can remove any periodic activity from drivers and change to interrupt or event-driven designs. If eliminating periodic activity is not possible, Windows timer coalescing helps the periodic activity become more efficient. For more information, see "**KeSetCoalescableTimer**" in the Windows Driver Kit (WDK).

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 is used to set the period when a timer should periodically expire. In many situations existing calls to **KeSetTimerEx** can easily be replaced 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. For more information, see "**KeSetTimerEx**" in the WDK.

Developers 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.

An alternative is to 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. For more information, see "*IoTimer*" in the WDK.

Timer Coalescing within a WDF Driver

Windows timer coalescing is available in the Windows Driver Framework (WDF) beginning with version 1.9. Developers 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

Developers can take advantage of Windows timer coalescing in applications and services by using the new user-mode **SetWaitableTimerEx** function. However, similar to device drivers, developers should first determine whether they can remove any periodic activity from applications and services and change to event-driven designs. If periodic activity cannot be eliminated, Windows timer coalescing helps the periodic activity become more efficient. For more information, see "**SetWaitableTimerEx** Function" on the MSDN website.

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 is used to set the period when a timer should periodically expire. In many situations it is easy to replace existing calls to **SetWaitableTimer** with calls to **SetWaitableTimerEx**. **SetWaitableTimerEx** has two new parameters: *WakeContext* and *TolerableDelay*. Use the *WakeContext* parameter only when setting 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. For more information, see "**SetWaitableTimerEx** Function" on the MSDN website.

Developers 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

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

When developers use the Windows timer coalescing functions, they 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.

Conclusion

Modern processors are adept at reducing power use when idle, and advances such as Intelligent Timer Tick Distribution enable further power reductions. Applications can allow systems to reduce power consumption by using timer resolution appropriately and can enhance power savings by using timer coalescing. By using timer coalescing, developers also facilitate future power saving advances by increasing idle periods.

Resources

MSDN:

SetWaitableTimerEx Function
http://msdn.microsoft.com/en-us/library/dd405521(VS.85).aspx

timeBeginPeriod Function
http://msdn.microsoft.com/en-us/library/dd757624%28VS.85%29.aspx

Windows Driver Kit (WDK):

IoTimer
http://msdn.microsoft.com/en-us/library/ff550381.aspx

KeSetCoalescableTimer
http://msdn.microsoft.com/en-us/library/ff553249(VS.85).aspx

KeSetTimerEx
http://msdn.microsoft.com/en-us/library/ff553292(VS.85).aspx

WHDC:

Energy Smart Software
http://www.microsoft.com/whdc/system/pnppwr/powermgmt/Energy-Smart_SW.mspx

The Science of Sleep
http://www.microsoft.com/whdc/system/pnppwr/powermgmt/Science-Sleep.mspx