

Fine-Grained Application Analysis for Energy-Aware Computing

This white paper describes how software “idle” behavior can have a negative impact on battery life. It also explains how to find the root cause of an application behavior that impacts battery life and the steps necessary to mitigate it. This document has been written for ISVs, OEMs, ODMs, the technical press, and technical enthusiasts.

Table of Contents

The Importance of Battery Life	1
Tools Introduction	1
Intel® Power Checker	1
Windows ⁵ Powercfg	1
Battery Life Analyzer (BLA)	1
Windows Performance Analyzer (xperf)	2
Environment Setup	2
Hardware Description	2
Software Description	2
Evaluation Experiment	2
Deep Analysis with Intel Power Checker	3
Deep Analysis with Battery Life Analyzer (BLA)	4
Deep Analysis with Windows Powercfg	4
Deep Analysis with Windows Performance Analyzer	5
Summary	8
References	8
About the Author	8

The Importance of Battery Life

Software plays an important role in battery life. The OS, firmware, drivers, and all small components are typically optimized to give better performance and energy efficiency. As the notebook PC (and smaller form factor devices, including tablets and smart phones) become pervasive compute platforms, battery life is becoming increasingly important, particularly with regard to standby or idle time.

In addition, as hardware power states become more sensitive, software must be well behaved at idle so it doesn’t needlessly wake components, which would limit battery life. Several case studies presented here show how software “idle” behavior can have a negative impact in this area on Windows⁵-based systems.

Tools Introduction

We used four easily available developer tools to mitigate the behavior of the application, as described in this section.

Intel® Power Checker

Intel® Power Checker can quickly assess idle power efficiency, platform timer tick, and power-aware behavior for applications that run on mobile platforms using the Intel® Core™ processor family or Intel® Atom™ processor. It is typically used as the first step of power analysis. It can also be used as cloud tool for developers to track power efficiency during the development cycle. Intel Power Checker also helps ISVs and OEMs create customized software assessment reports on power efficiency.

Windows⁵ Powercfg

Windows⁵ Powercfg is a command-line utility used to control power settings. It uses Event Tracing for Windows (ETW) to profile systems. This tool was shipped as a separate piece of software to control power settings in Windows XP and Windows Vista⁵, but it shipped as part of Windows 7. Users can use Windows

Powercfg to view and modify power plans and settings such as standby time, wake timer, and power schemes.

Powercfg-energy (with energy option) analyzes common energy-efficiency and battery life problems, such as platform timer-tick setting changes, changes to timers by application processes or DLLs, processor utilization per process, and power-management settings in hardware and OSs. Administrator permission is required to run Windows Powercfg.

More information on the use of Windows Powercfg can be found at [http://technet.microsoft.com/en-us/library/cc748940\(ws.10\).aspx](http://technet.microsoft.com/en-us/library/cc748940(ws.10).aspx).

Battery Life Analyzer (BLA)

This simple tool written by Intel monitors various software and hardware activities that affect battery life on Windows-based systems. BLA supports Intel® 5 Series Mobile Chipset, the Intel® Core™ i7, Core™ i5, and Core™ i3 processors, and 2nd generation Intel® Core™ i7, Core™ i5, and Core™ i3 processors. BLA can identify misbehaving drivers, processes, and

hardware that prevent the platform from entering low power states. It also provides power impact estimations based on an algorithm for each offending hardware and software component.

Windows Performance Analyzer (xperf)

Windows Performance Analyzer (WPA), also known as xperf, is a set of performance monitoring tools used to produce in-depth performance profiles of Microsoft Windows OSs and applications.

Environment Setup

Hardware Description

- Processor: 2nd generation Intel Core i7 mobile processor
- Memory: 4 GB DDR3
- Storage: 80 GB Intel® Solid State Drive
- Intel® Wi-Fi
- Realtek Audio^s

Software Description

- Microsoft Windows 7 Ultimate
- Intel® Wi-Fi driver
- Intel® HD Graphics driver
- Intel® Chipset driver

To get started, we made the following setting changes to a fresh installation of Windows 7 to eliminate certain effects of overhead in the power measurements:

- **Disable screen saver** to eliminate variability because of screen changes
- **Disable display dimming** to avoid variability in display power usage
- **Disable system protection**, auto-backup, and auto-update features to avoid these services running in the background
- **Disable scheduled disk defragmentation** to avoid extraneous power usage from this source
- **Stop the Windows Search Indexer** to eliminate that source of run-to-run variability
- **Disable firewall and Windows Defender** to eliminate those sources of overhead during the experiment

Evaluation Experiment

The main focus of this paper is to accurately characterize the battery life impact of applications during their “idle” state. In this paper, application “idle” state is defined as a state where the application is launched (login is done if needed for connected applications). For applications that require network connectivity, power measurement is done while connected to the Internet. For the experiment, we randomly selected applications in daily use by typical industry users from the following categories:

- Media playback applications
- Messenger/chat applications
- Browser applications
- Anti-virus applications

During the first phase of our analysis, we selected top applications in the current consumer market from different categories and measured total power above baseline using the Fluke NetDAQ^s instrument. Figure 1 shows baseline “system idle” total platform power with a fresh Windows install (including drivers).

Figure 1 shows the overview of the “idle” battery usage over system idle. The Y-axis shows the percentage of power increase, while the X-axis shows different application categories. For example, if system idle is 7 watts and a media application takes 30 percent of power over baseline, we can surmise that this media application consumes 2.1 watts of power at “idle.”

Comparing Figure 2, which was taken from a presentation given at the [Intel Developer Forum 2010, San Francisco](#), we can interpret that the media application decreases system battery life at “idle” by approximately 75 minutes while not “active,” a clearly significant impact.

This paper studies why “idle” software impacts battery life and how small changes by developers can extend the battery life of target systems. From Figure 1 we selected for deeper analysis the worst performing application in terms of power usage at “idle.”

To perform the analysis, we downloaded a full version of the media application onto a system with a clean Windows 7 Ultimate

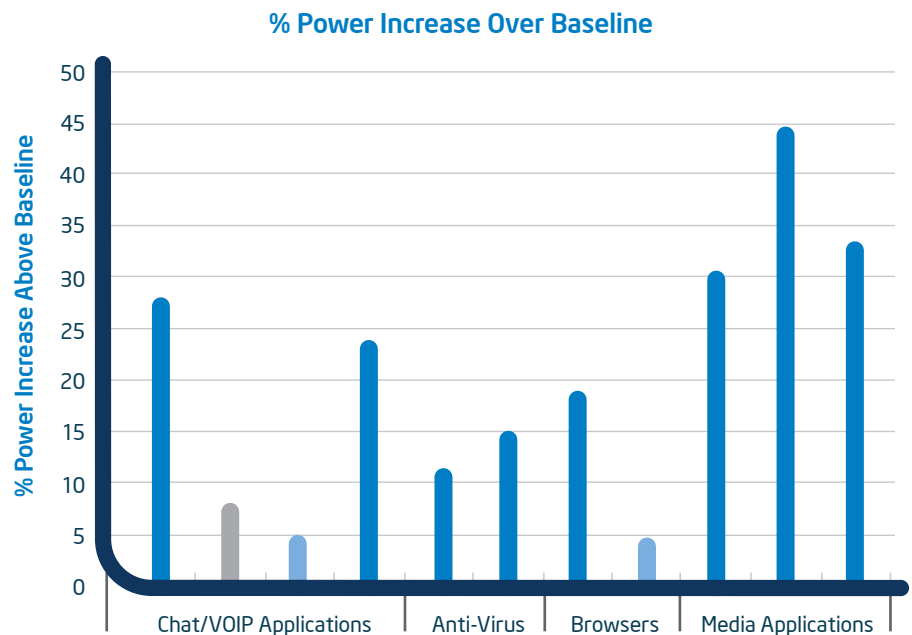


Figure 1. Idle power analysis.

install. We simulated a real-world scenario by opening the application in default windowed mode and let it run for 10 to 15 minutes. We collected total platform power utilization using the Fluke NetDAQ system for 10 minutes. Figure 3 shows the total platform power consumption over time between “clean system idle” and “media application idle.”

An application at “idle” must not act as an “active” workload; it must increase power usage only minimally, relative to system idle. To identify the reason for the increase in power usage, we used tools for root-cause analysis, as described below.

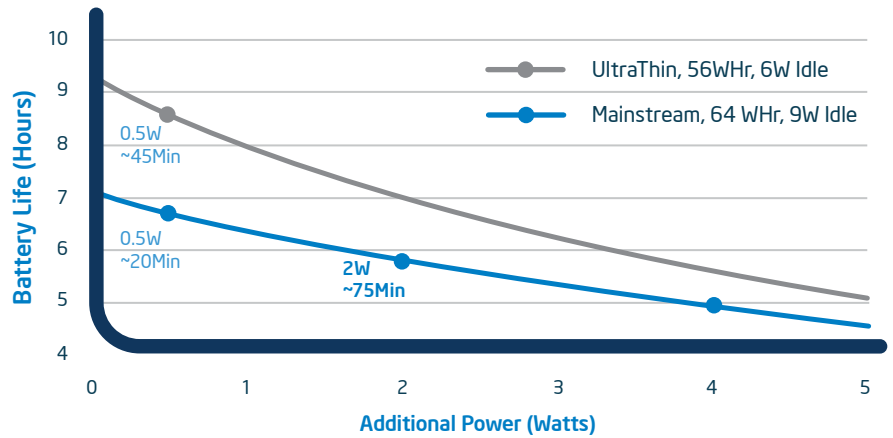


Figure 2. Intel Developer Forum reference battery life.

System “Idle” Platform Power vs. Media Application “idle” Power

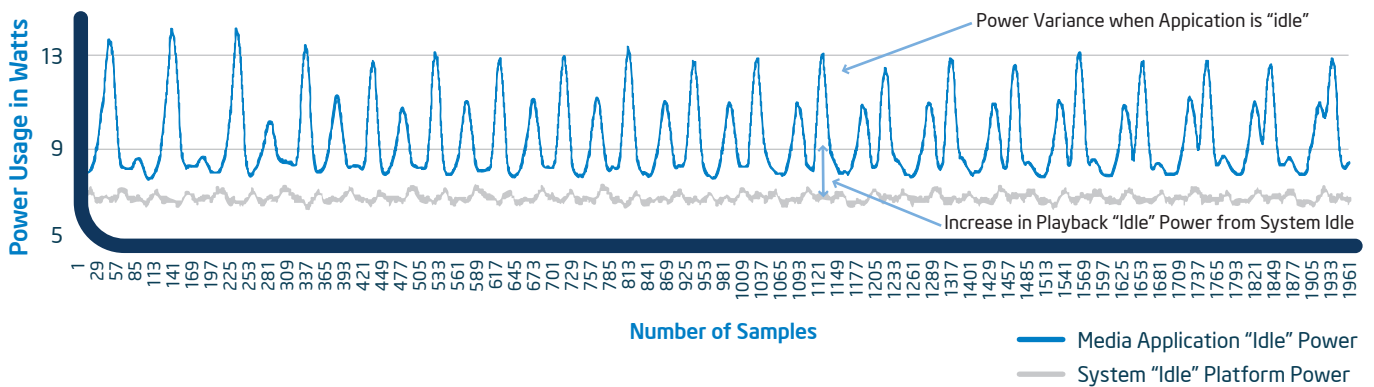


Figure 3. System “idle” platform power versus media application “idle” power.

Deep Analysis with Intel Power Checker

This simple application measures high level power usage by idle and active applications. The following steps provide high-level information from Intel Power Checker, as illustrated in Figure 4:

1. Enter name of application (e.g., Media Playback)
2. Select “Measure Power Awareness”
3. Select “Generate Report File”
4. Select “Launch Application”

Once the process is completed, Intel Power Checker generates a report showing the change in C-state, timer tick, and other measurements between baseline and application workload. The results in Figure 5 show that this media application set timer tick to 1.25msec from 15.6 msec.

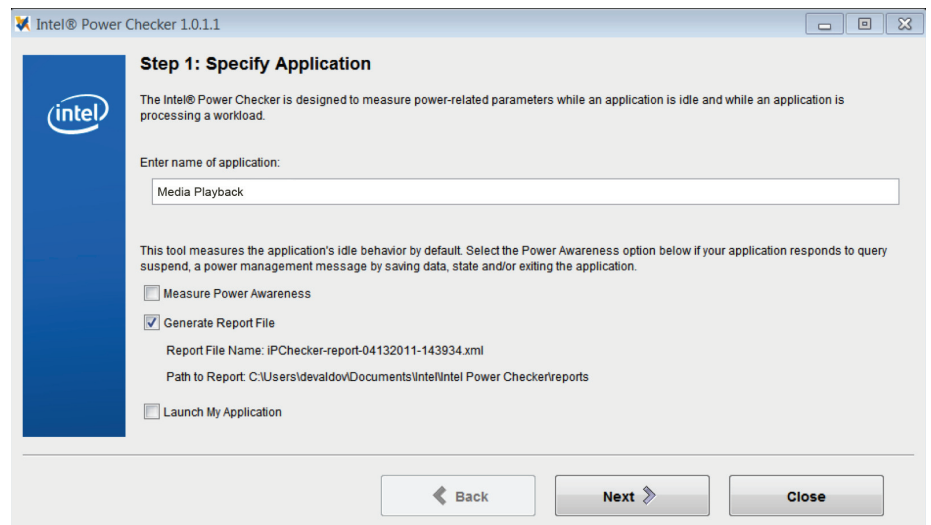


Figure 4. Intel® Power Checker initiation screen.

Deep Analysis with Battery Life Analyzer (BLA)

BLA is a very simple, lightweight tool written by Intel to find high-level issues related to hardware and software. BLA indicates misbehaving drivers, processes, and hardware that prevent the platform from entering low-power states.

We collected C-state data with BLA in "system idle" and "application idle" periods. Figure 6 shows the impact of the "idle" application on C-state. When more time is spent in deeper C-states, more power is saved, but more time is required for the processor to return to its full-power operating state. For more information on C-state, see <http://origin-software.intel.com/en-us/blogs/2008/03/27/update-c-states-c-states-and-even-more-c-states/>.

Deep Analysis with Windows Powercfg

C-state data from BLA indicated that processor wakeup frequently causes more C-state transitions. To analyze the cause of C-state package residency, we used the Windows Powercfg command-line utility to collect a 60-second trace. Figure 7 shows the output of the Powercfg tool.

In Figure 7, the Powercfg utility shows that the media application is changing timer tick resolution to 1.25 msec from the system default of 15.6 msec, even at "idle" when there is no activity. Changing the timer tick to high granularity causes more frequent wake-ups; a timer tick setting of 15.6 msec creates 64 periodic wake-ups per second, and a 1.25 msec setting generates 800 periodic calls per second. That figure represents 10 times more calls per second than at "idle."

To check the significance of timer tick settings on actual power usage, we changed the timer tick setting to 15.6 msec at "idle." Figure 9 shows the change in power numbers. Setting the timer tick back to the system default creates significant gains in power savings. Average

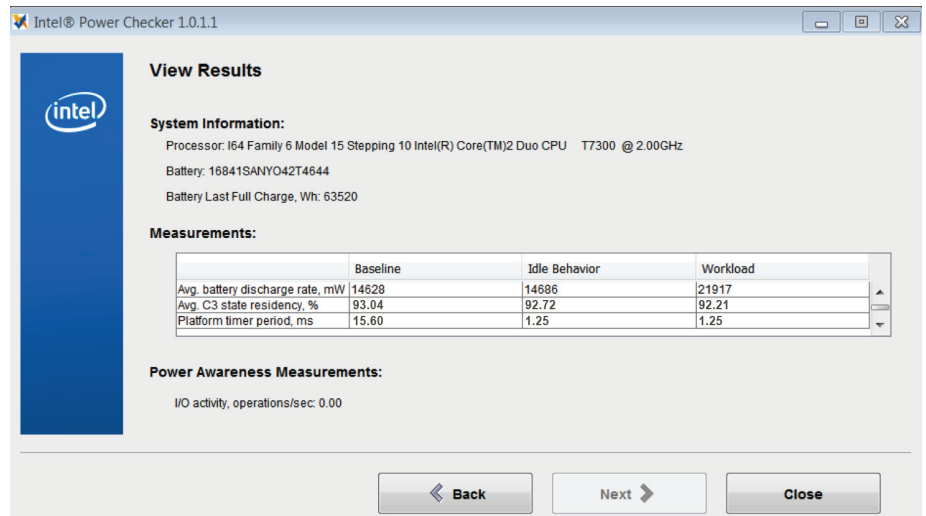


Figure 5. Intel® Power Checker results screen.

power increases for "application" idle decreases from 44 percent (default timer tick) to 16 percent (timer tick setting of 15.6).

Next we drilled down to perform more detailed analysis and to look at other issues that cause power consumption, using BLA and Windows Performance Analyzer to find the root causes. BLA gives details of periodic activity including timer tick when the application is running idle, as shown in Table 1.

System file **hal.dll** signifies calls due to timer tick activity, and changing the timer tick setting to 1.25 msec increases calls per second from 64 to 800. Graphics activity was another source of power usage shown by BLA for this media application. Table 2 shows the threshold of an application at "idle" to have minimal impact on power due to interrupts.

The first level of analysis as described in previous sections determines the actual power impact due to application "idle,"

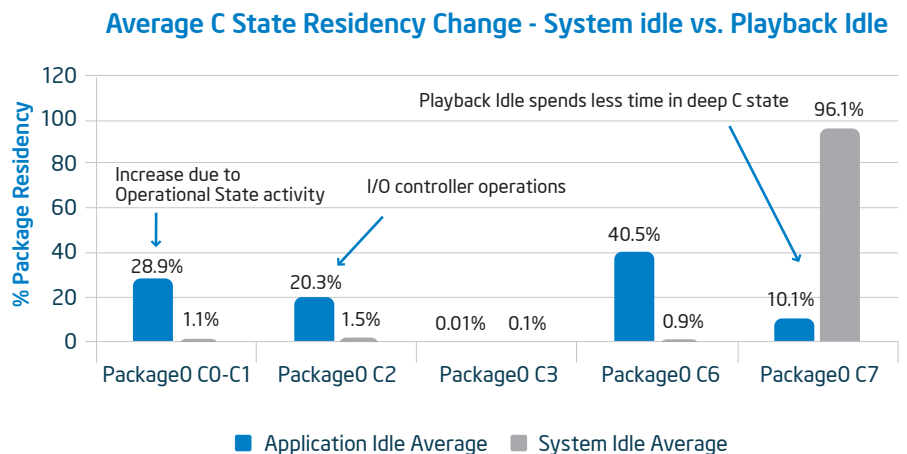


Figure 6. Package C-state residency: system idle versus playback idle

Platform Timer Resolution: Platform Timer Resolution
 The default platform timer resolution is 15.6ms (15625000ns) and should be used whenever the system is idle. If the timer resolution is increased, processor power management technologies may not be effective. The timer resolution may be increased due to multimedia playback or graphical animations.
 Current Timer Resolution (100ns units) **12500**
 Maximum Timer Period (100ns) **156001**

Platform Timer Resolution: Outstanding Timer Request
 A program or service has requested a time resolution smaller than the platform maximum timer resolution.
 Requested Period **20000**
 Requesting Process ID **2792**
 Requesting Process Path **\\Device\\HarddiskVolume2\\Program Files (x86)\\Media\\MediaPlayer\\Player.exe**

Platform Timer Resolution: Timer Request Stack
 The stack of modules responsible for the lowest platform time setting in this process.
 Requested Period **20000**
 Requesting Process ID **2792**
 Requesting Process Path **\\Device\\HarddiskVolume2\\Program Files (x86)\\Media\\MediaPlayer\\Player.exe**
 Calling Module Stack **\\Device\\HarddiskVolume2\\Windows\\SysWOW64\\ntdll.dll**
 \\Device\\HarddiskVolume2\\Windows\\SysWOW64\\winmm.dll
 \\Device\\HarddiskVolume2\\Windows\\Program Files (x86)\\Media\\MediaPlayer\\Media.dll

Figure 7. Windows[®] Powercfg output.

change in timer tick from Windows default settings, and analysis of C-state package residency and various driver interrupts. For the next level of analysis, we use Windows Performance Analyzer (xperf) to determine whether synchronization API usage causes interrupts.

Deep Analysis with Windows Performance Analyzer
 We downloaded Windows Performance Analyzer and collected a CSwitches trace for 180 seconds on a media "idle" application using the following command line script:

```
run_xperf.cmd:
xperf -on diageasy
call wait 120
xperf -d NewTrace.etl

wait.cmd
@choice /D y /T %1% >NUL
```

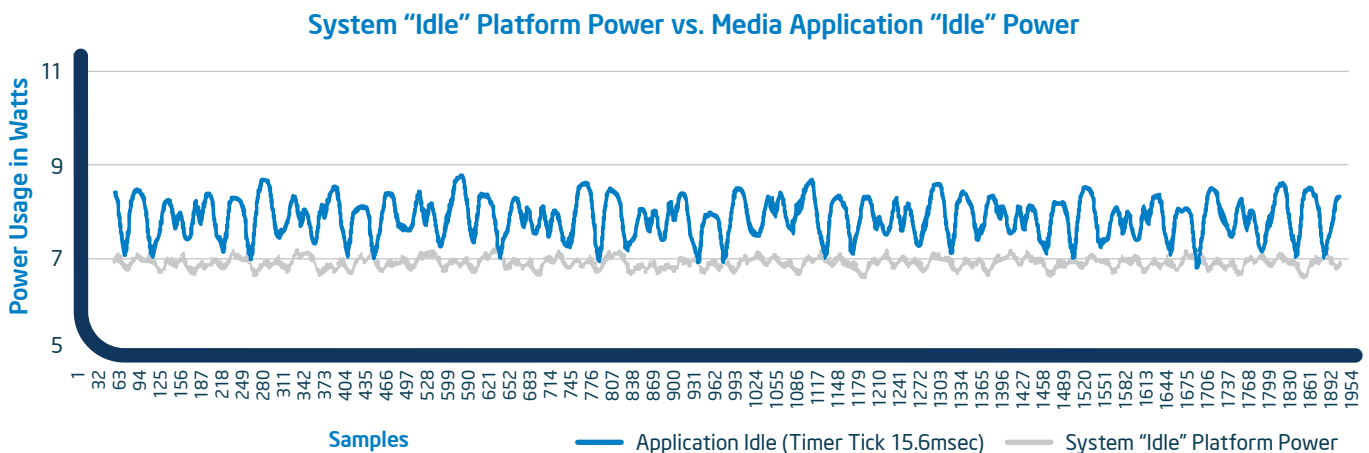


Figure 8. System "idle" versus media idle after overriding timer tick.

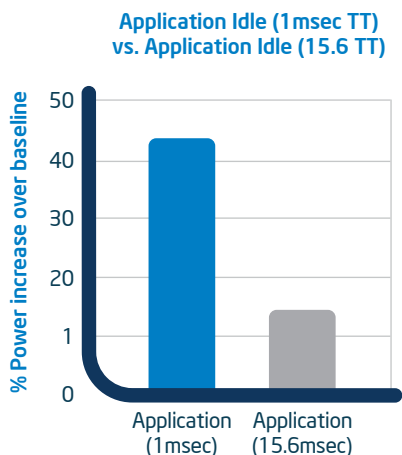


Figure 9. Application idle (1msec TT) versus application idle (15.6 TT).

This script collects the stack walk Windows trace for 180 seconds and outputs the file **newtrace.etl**. We opened that file in xperf and resolved the symbols by adding symbol path. Note that it is important to set the symbol paths before enabling “Load Symbols” and before opening a summary view. You can also modify the **_NT_SYMBOL_PATH** and **_NT_SYMCACHE_PATH** environment variables to make these changes permanent. The standard symbol path that includes Microsoft’s symbol server configuration is as follows:

```
_NT_SYMCACHE_PATH:C:\symbols
_NT_SYMBOL_PATH:
srv*c:\symbols*http://msdl.microsoft.com/download/symbols;
```

To add symbols from your own builds, add **C:\apps\directyr\bin** to **_NT_SYMBOL_PATH**. As with all Windows paths, the symbol path uses semicolons (;) as separators. Figure 10 shows the high level view using xperf for the media playback application.

Table 1. Periodic activity during idle application running time.

IMAGE NAME	SYSTEM IDLE CALLS/SEC	PLAYBACK IDLE CALLS/SEC	REASON
hal.dll	64.1	800	Due to timer tick setting change
dxgkrnl.sys	1	290	Due to graphics activity at “idle”
USBPORT.SYS	10	18	
ataport.SYS	3.0	4.2	

Table 2. Application threshold at idle impact on power due to interrupts.

DRIVER	INTERRUPT SOURCE	TYPICAL INTERRUPTS/SEC @ IDLE
ACPI.sys	Motherboard	<1
dxgkrnl.sys	Graphics	<10 (depends on activity)
hal.dll	Timer tick	64
i8042prt.sys	PS/2 KB/Mouse	0
msahci.sys	SATA (HDD/DVD)	<10 (depends on activity)
ndis.sys	Network (GbE/Wi-Fi)	<40 (Wi-Fi, associated, no traffic)
USBPORT.SYS	USB (UHCI/EHCI)	0

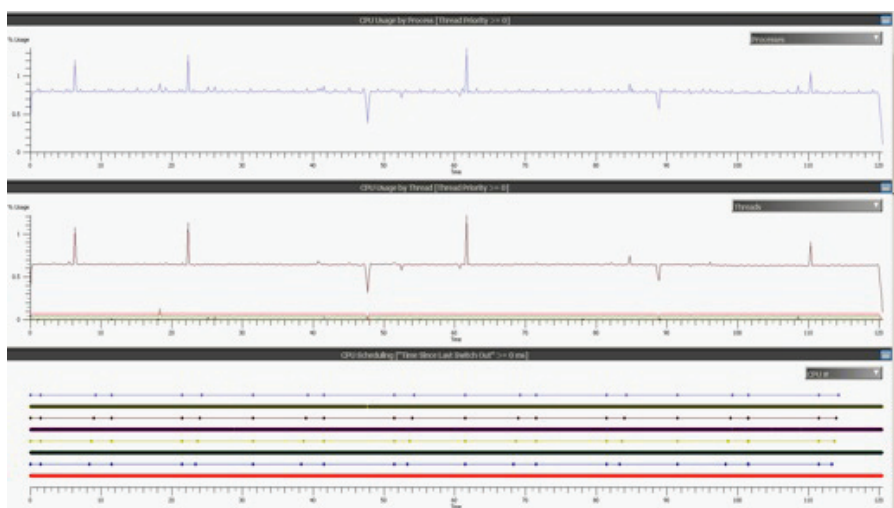


Figure 10. High-level view for Windows^s xperf.

The next step is to drill down to the issues that are causing frequent processor wake-ups. To do so, right-click **CPU Scheduling** and click **Select View**. Right-click again and click to view the summary table. Remember to have **Load Symbols** selected from the menu.

Figure 11 shows context switches occurring at "idle" when no audio or video is playing. One of the main causes of these context switches is timer tick activity. Another reason was the media application calling a sleep function every 10 msec, as shown in Figure 12.

Line	NewProcess	NewThreadId	SwitchInTime (s)	NewThreadStack	Count	Sum:TimeSinceLast (s)
1	Idle (0)	0			122,3...	0.000
2	Media (612)				43,426	2,011,500,730.503
3		1,440			10,117	179,774,977.735
4		2,240			10,890	172,590,291.202
5		1,228			6,093	100,356,069.006
6		1,600			3,627	100,056,672.636
7		2,524			3,623	100,012,743.511
8		1,864			101	179,983,077.707
9		952			164	179,304,006.400
10		2,776			92	178,100,825.135
11		2,340			91	178,103,972.607

Figure 11. Context Switches view from xperf.

Visualizing the code, we see that this media application had several sleep functions in a loop:

```

for(;;){
    if(.. check some condition) {
        ..do something...
        SetEvent()
    }
    Sleep(10); //wait for next tick
}
    
```

A better implementation would be the following:

```

for(;;) { WaitForSingleObject();
.... Do something #1 ...
}
for(;;) { WaitForSingleObject();
.... Do something #2 ...
}
    
```

Line	NewProcess	NewThreadId	SwitchInTime (s)	NewThreadStack	Count	Sum:TimeSinceLast (s)
1	Idle (0)	0			122,3...	0.000
2	Media (612)				43,426	2,011,500,730.503
3		1,440			10,117	179,774,977.735
4		39,097,973,570	[Root]		1	10,000,354
5		63,307,555,507	[Root]		1	10,073,557
6		174,285,952,999	[Root]		1	10,072,625
7		70,247,385,005	[Root]		1	10,068,892
8			KernelBase.dll!Sleep		1	10,068,892
9			KernelBase.dll!SleepEx		1	10,068,892
10			ntdll.dll!ZwDelayExecution		1	10,068,892
11			ntdll.dll!DbgInitializeThunk		1	10,068,892
12			ntdll.dll!??_FNODBFM_?string'		1	10,068,892
13			wow64.dll!Wow64DbgInitialize		1	10,068,892
14			wow64.dll!RunCpuSimulation		1	10,068,892
15			wow64cpu.dll!Thunk2ArgNpNpReloadState		1	10,068,892
16			wow64cpu.dll!CpuSyscallStub		1	10,068,892
17			ntkrnlmp.exe!KGSystemServiceCopyEnd		1	10,068,892
18			ntkrnlmp.exe!KdDelayExecution		1	10,068,892
19			ntkrnlmp.exe!KdDelayExecutionThread		1	10,068,892
20			ntkrnlmp.exe!KdCommitThreadWait		1	10,068,892
21			ntkrnlmp.exe!KdSwapContext		1	10,068,892
22			ntkrnlmp.exe!SwapContext_PatchRouter		1	10,068,892
23					1	10,068,892
24			[Root]		1	10,056,763
25			KernelBase.dll!Sleep		1	10,056,763
26			KernelBase.dll!SleepEx		1	10,056,763
27			ntdll.dll!ZwDelayExecution		1	10,056,763
28			ntdll.dll!DbgInitializeThunk		1	10,056,763
29			ntdll.dll!??_FNODBFM_?string'		1	10,056,763
30			wow64.dll!Wow64DbgInitialize		1	10,056,763
31			wow64.dll!RunCpuSimulation		1	10,056,763
32			wow64cpu.dll!Thunk2ArgNpNpReloadState		1	10,056,763

Figure 12. Sleep function calls from a media application.

Fine-Grained Application Analysis for Energy-Aware Computing

Summary

Software has a huge impact on “idle” and “active” battery life of mobile systems. Small changes in code can help bring the power level of “application idle” closer to that of “platform idle.” Setting timer tick to a finer granularity, even at “idle” where performance and quality is not a concern, is not a good programming practice. Use of Windows Coalescing APIs, event polling, and keeping default timer tick settings can help to extend the battery life of mobile systems.

References

Intel® Developer Forum 2010: http://download.intel.com/technology/pdf/Impact_of_idle_SW_on_battery_life.pdf

Intel® Power Checker: <http://www.intel.com/partner/sat/pe>

Microsoft Timer Coalescing: <http://download.microsoft.com/download/9/C/5/9C5B2167-8017-4BAE-9FDE-D599BAC8184A/TimerCoal.docx>

Microsoft Powercfg: [http://technet.microsoft.com/en-us/library/cc748940\(ws.10\).aspx](http://technet.microsoft.com/en-us/library/cc748940(ws.10).aspx)



About the Author

Manuj Sabharwal is a software engineer in the Software Solutions Group at Intel. Manuj has been involved in exploring power enhancement opportunities for idle and active software workloads. He has significant research experience in power efficiency and has delivered tutorials and technical sessions in the industry. He also works on enabling client platforms through software optimization techniques.

Intel® compilers, associated libraries, and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel® microarchitecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the “Intel® Compiler User and Reference Guides” under “Compiler Options.” Many library routines that are part of Intel compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel compilers, associated libraries, and associated development tools may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20101101

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked “reserved” or “undefined.” Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: <http://www.intel.com/design/literature.htm>

Results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.

Intel, the Intel logo, Atom, Core, and Xeon are trademarks of Intel Corporation in the U.S. and other countries.

®Other names and brands may be claimed as the property of others.

Copyright© 2011 Intel Corporation. All rights reserved.

0411/JH/MESH/PDF

325365-001US

