



PeekMessage: Optimizing Applications for Extended Battery Life

by Dale Taylor

May 2005

The information contained in this document is provided for informational purposes only and represents the current view of Intel Corporation Intel on the date of publication. Intel makes no commitment to update the information contained in this document, and Intel reserves the right to make changes at any time, without notice.

DISCLAIMER. THIS DOCUMENT AND ALL INFORMATION CONTAINED HEREIN IS PROVIDED AS IS. INTEL MAKES NO REPRESENTATIONS OF ANY KIND WITH RESPECT TO PRODUCTS REFERENCED HEREIN, WHETHER SUCH PRODUCTS ARE THOSE OF INTEL OR THIRD PARTIES. INTEL EXPRESSLY DISCLAIMS ANY AND ALL WARRANTIES, IMPLIED OR EXPRESS, INCLUDING WITHOUT LIMITATION, ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, AND ANY WARRANTY ARISING OUT OF THE INFORMATION CONTAINED HEREIN, INCLUDING WITHOUT LIMITATION, ANY PRODUCTS, SPECIFICATIONS, OR OTHER MATERIALS REFERENCED HEREIN. INTEL DOES NOT WARRANT THAT THIS DOCUMENT OR THE INFORMATION CONTAINED HEREIN IS FREE FROM ERRORS, OR THAT ANY PRODUCTS OR OTHER TECHNOLOGY DEVELOPED IN CONFORMANCE WITH THIS DOCUMENT WILL PERFORM IN THE INTENDED MANNER, OR WILL BE FREE FROM INFRINGEMENT OF THIRD PARTY PROPRIETARY RIGHTS, AND INTEL DISCLAIMS ALL LIABILITY THEREFOR.

INTEL DOES NOT WARRANT THAT ANY PRODUCT REFERENCED HEREIN OR ANY PRODUCT OR TECHNOLOGY DEVELOPED IN RELIANCE UPON THIS DOCUMENT, IN WHOLE OR IN PART, WILL BE SUFFICIENT, ACCURATE, RELIABLE, COMPLETE, FREE FROM DEFECTS OR SAFE FOR ITS INTENDED PURPOSE, AND HEREBY DISCLAIMS ALL LIABILITIES THEREFOR. ANY PERSON MAKING, USING OR SELLING SUCH PRODUCT OR TECHNOLOGY DOES SO AT HIS OR HER OWN RISK.

Licenses may be required. Intel and others may have patents or pending patent applications, trademarks, copyrights or other intellectual proprietary rights covering subject matter contained or described in this document. No license, express, implied, by estoppels or otherwise, to any intellectual property rights of Intel or any other party is granted herein. It is your responsibility to seek licenses for such intellectual property rights from Intel and others where appropriate.

Limited License Grant. Intel hereby grants you a limited copyright license to copy this document for your use and internal distribution only. You may not distribute this document externally, in whole or in part, to any other person or entity.

LIMITED LIABILITY. IN NO EVENT SHALL INTEL HAVE ANY LIABILITY TO YOU OR TO ANY OTHER THIRD PARTY, FOR ANY LOST PROFITS, LOST DATA, LOSS OF USE OR COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, OR FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF YOUR USE OF THIS DOCUMENT OR RELIANCE UPON THE INFORMATION CONTAINED HEREIN, UNDER ANY CAUSE OF ACTION OR THEORY OF LIABILITY, AND IRRESPECTIVE OF WHETHER INTEL HAS ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES. THESE LIMITATIONS SHALL APPLY NOTWITHSTANDING THE FAILURE OF THE ESSENTIAL PURPOSE OF ANY LIMITED REMEDY.

Intel, the Intel logo, Pentium, Intel Xeon, and VTune are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

Copyright © 2005 Intel Corporation

Introduction

When discussing optimization techniques for applications running on battery powered devices, some application developers doubt or do not understand the need to be concerned with this issue. To better understand this need, some detailed research including metrics on a specific example would be of great value. This document focuses on the use of PeekMessage calls and its impact on battery life, if any. It also explains why this is possible, discusses the reasons, and provides examples showing alternative coding samples and the kind of improvements that can be expected when you account for how PeekMessage affects Windows*. A sample program along with the Power Evaluation Tool from the Intel Corporation Mobilized Software Initiative Technical Developers Kit (MSI TDK) is used to measure the actual changes when switching from PeekMessage to GetMessage calls.

PeekMessage

The MSDN documentation contains the following discussion concerning PeekMessage:

An application should use a PeekMessage() loop for as little time as possible. To be compatible with battery-powered computers and to optimize system performance, every Windows-based application should inform Windows that it is idle as soon and as often as possible¹.

While the MSDN authors are aware that PeekMessage can cause problems associated with power consumption, just how and why requires further research. As additional support of this side effect of using PeekMessage, we find the following quote from another MSDN article:

A power-friendly PeekMessage() loop exits when background processing is complete because, while an application is in a PeekMessage() loop, Windows cannot go idle².

With PeekMessage being heavily used for years by many legacy applications as the key component of a message loop, it becomes important to account for power issues in the applications. Basically, the PeekMessage call ties up the system by keeping it busy checking for a message. The analogy you can use here while discussing PeekMessage is that of children when on a road trip. They tend to continually ask you “Are we there yet?” By peeking for a message, you continually ask Windows if there’s a message for you. You can still use PeekMessage, but you need to be aware of the power implications and enable a proper idle state when the program is in fact idle. An application can be written to be power friendly, allow windows to go idle, and also monitor roughly how long the application has been idling without processing a message. Given the difficulties of using PeekMessage properly, alternative methods of dealing with a potential PeekMessage situation have been devised.

One way to directly control how PeekMessage is used in your application is to build your own message loop. By ensuring you cover the essential elements as found in Microsoft Foundations Classes* (MFC) main message loop, your application can directly control the message pump. Following is a short pseudo code sample that is compatible with MFC.

```

while ( bDoingBackgroundProcessing )
{
    MSG msg;
    while ( ::PeekMessage( &msg, NULL, 0, 0, PM_NOREMOVE ) )
    {
        if ( !PumpMessage( ) )
        {
            bDoingBackgroundProcessing = FALSE;
            ::PostQuitMessage( );
            break;
        }
    }
    // let MFC do its idle processing
    LONG lIdle = 0;
    while ( AfxGetApp()->OnIdle(lIdle++ ) )
        ;
    // Perform some background processing here
    // using another call to OnIdle
}

```

Dissecting this code, you see that as long as there is idle processing going on, PeekMessage will be called. The PumpMessage is used to perform normal message translation and dispatching. The problem in using this code is with the PumpMessage call. While the source can be found in ThrdCore.cpp, it is officially undocumented and thus could be subject to change in future releases (although unlikely at this point).

Now, a method of managing small chunks of time and work in a manner consistent with methods that will help Windows know when to sleep and conserve power is available. If you want to determine when an application should sleep and enable power savings directly, use the WaitMessage call. WaitMessage will allow your application to directly control when it's put to sleep. Back to the analogy of a road trip with children, when you have the children using PeekMessage methods, they are continually hounding you with "Are we there yet?" questions. Using WaitMessage is just like telling the kids "Go ahead and take a nap, I'll wake you when we get there." Using WaitMessage frees the system to focus on other tasks, just like you can when driving and your kids go to sleep. Here are a few straightforward lines of pseudo code that clearly show how this can be done.

```

if ( PeekMessage(...) != NULL)
    // IF there is a message, translate & dispatch it
else if (there is background processing to do)
    // do your background processing
else
    // No background processing, no messages waiting - go to sleep
    WaitMessage();

```

Writing your application to manage itself, as shown in the above pseudo code, will help improve the power usage on battery powered systems.

Evaluating Sample Apps Using Intel's Battery Monitoring Tools

It's clear from the MSDN documentation and the above examples, that Windows internally won't let an application sleep while it makes PeekMessage calls. To validate this research, some real metrics were obtained to support the findings. A small application was developed to simulate the possible effects of these calls. This application uses either PeekMessage or GetMessage calls for a timed duration to monitor an input window. Different versions of the sample application were built, which make the different calls and run for different lengths of time. To monitor the effects this program has on the power usage of a system, the Power Evaluation Tool was developed internally by the Mobility Enabling group at Intel. This tool is used to launch the different builds and determine the battery usage for the different tests.

These are contrived tests where one build does only PeekMessage calls and the other does only GetMessage calls. In a real world application, there would be some mixing of these cases. The extreme nature of the test case helps highlight the differences in the message calls. To better understand the results presented below, you need to know a little about the Power Evaluation Tool. The Mode setting in the table below refers to the two modes supported by the Power Evaluation Tool. Mode 1 only runs the application until it self-terminates (the application includes a timing loop). Mode 2 runs the application and then monitors the background for a matching time period to enable a comparison of the background usage to the background AND the application usage combined. Mode 2 tests can thus provide some additional insight into power usage. These tests were run on an IBM T41* with a new battery. Each test started with the battery recharged to 96-98%. The T41 has an Intel IT build installed. See the Appendix for additional details concerning the IT build, which includes the monitoring software, support tools, a virus scanner and so on.

Test, Power Eval settings and duration	Power Consumed Process + Background	Power Consumed Background	Power Consumed Process
PeekMessage, Mode 1, 15 minute run	3171 mwh	NA	NA
PeekMessage, Mode 2, 15 minute run	3123 mwh	2678 mwh	445 mwh
GetMessage, Mode 1, 15 minute run	2722 mwh	NA	NA
GetMessage, Mode 2, 15 minute run	2698 mwh	2636 mwh	61 mwh

Table 1 Peek vs. Get Results #1

While looking at this data observe that the power consumed by the process is much higher in the Peek test vs. the Get tests (be sure to compare like tests). About 7x more power is consumed by the process. This can be seen in the right most column where the power consumed by the process alone is shown. This shows that the process power requirements are much higher in the peek tests.

This leads to the next set of data, and the conclusion that the average power and net power consumed contains more useful data since the power consumed is averaged over the time the test ran, in essence providing a power "drain" rate.

Test, Power Eval settings and duration	Ave Power Consumed Process + Background	Ave Power Consumed Background	Net Ave Power Consumed Process
PeekMessage, Mode1, 15 minute run	12.3 w	NA	NA
PeekMessage, Mode 2, 15 minute run	11.4 w	9.8 w	1.6 w
GetMessage, Mode 1, 15 minute run	9.6 w	NA	NA
GetMessage, Mode 2, 15 minute run	9.6 w	9.6 w	0.1 w

Table 2 Peek vs. Get Results #2

While looking at this data, observe that while the background power consumed remains the same across tests, the power consumed by the process is roughly 15x higher for the Peek tests. Here it's clear that the overall power consumed by the Peek tests is greater than that of the *Get* tests.

Test, Power Eval settings and duration	Ave % total processor time	Power mode OS set to use	Processor Speed	Average power Consumed
PeekMessage, Mode 1, 15 min run	100 %	Slow	589 MHz	0 %
PeekMessage, Mode 2, 15 min run	100 %	Slow	589 MHz	0 %
GetMessage, Mode 1, 15 min run	1%	Slow	589 MHz	1 %
GetMessage, Mode 2, 15 min run	1%	Slow	589 MHz	2 %

Table 3 Peek vs. Get Results #3

This data confirms that when running the *Peek* test, the system is unable to sleep. When running the *Get* test, the system spent nearly all of its time sleeping.

All of the above tests were run with the laptop in Intel's IT default battery mode of SLOW. To enable a more realistic set of test results, the following tests were done in high speed or adaptive mode. Only the longer tests are used. While the five minute tests were interesting to review, they don't tend to provide any useful additional data than the longer tests. Also, the longer tests work much better due to the design of the Power Evaluation Tool. The result tables with the shorter test are provided as additional backup material and are available at the end of this document.

Test, Power Eval settings and duration	Ave % total processor time	Power mode OS set to use	Processor Speed	Average power Consumed
GetMessage, Mode 1, 15 min run	1%	High speed	1594 MHz	15.3 W
GetMessage, Mode 1, 15 min run	3 %	Adaptive	589 MHz	13.0 W
GetMessage, Mode 2, 15 min run	2 %	Adaptive	589 MHz	12.3 W

Test, Power Eval settings and duration	Ave % total processor time	Power mode OS set to use	Processor Speed	Average power Consumed
PeekMessage, Mode 1, 15 min run	100 %	High speed	1594 MHz	29.7 W
PeekMessage, Mode 1, 15 min run	100 %	Adaptive	1592 MHz	27.9 W
PeekMessage, Mode 2, 15 min run	100 %	Adaptive	1592 MHz	28.5 W

These tests amplify the results even further. In these tests, the power consumed overall by the entire system is roughly double or higher on the system running the Peek tests. This clearly demonstrates that tuning your mobile applications to account for the PeekMessage problem can make a large difference in the power consumed by the application.

Conclusion

Application code needs to reflect an awareness of the impact a design can have on the consumption of system power. Excessive use of PeekMessage keeps the system from being able to sleep. PeekMessages should be used properly so that the system can best manage the power resources. Using the above methods for GetMessage and WaitMessage will enable your application to be more power friendly and help support our objective of increased power savings and performance.

Additional Resources

- For access to the Mobilized Software Resource Kit and Power Evaluation Tool refer to: <http://intelmktg.com/sdd/>
- [Not Off the Hook – Why App developers are an important link in the power-management chain](#)
- [DVD/CD Rendering: Optimizing for Power on Mobile Platforms](#)

About the Author



Dale is an Application Engineer working on Mobilized Software. He joined Intel in 1999, working on the LanDesk management software project, and then as a performance tuning engineer for the Software Solutions Group. Dale has been using his 22 years of programming and development experience to help tune and optimize customers so their products take advantage of Intel's latest mobility features. After programming for a government contractor, he spent 12 years programming in assembly and C as a developer of WordPerfect; followed by 4 years working for an independent startup.

Appendix A

IT Build information for Intel Corporation Intel IT systems come pre-installed with a set of software tools deemed necessary by the Intel IT department to ensure corporate security and compliance to various licensing agreements. The software includes virus scanners and a Terminus Security agent. The scanners and monitoring software are always running and add an additional load on the system.

Peek vs. Get Results #1 — Additional Results

Test Name	Test, Power Eval settings and duration	Power Consumed Process + Background	Power Consumed Background	Power Consumed Process
M1 Get	GetMessage, Mode 1, 5 minute run	1254 mwh	NA	NA
M2 Get	GetMessage, Mode 2, 5 minute run	1737 mwh	1731 mwh	7 mwh
M1 Get 15	GetMessage, Mode 1, 15 minute run	2722 mwh	NA	NA
M2 Get 15	GetMessage, Mode 2, 15 minute run	2698 mwh	2636 mwh	61 mwh
M1 Peek	PeekMessage, Mode 1, 5 minute run	1056 mwh	NA	NA
M2 Peek	PeekMessage, Mode 2, 5 minute run	1054 mwh	911 mwh	143 mwh
M1 Peek 15	PeekMessage, Mode 1, 15 minute run	3171 mwh	NA	NA
M2 Peek 15	PeekMessage, Mode 2, 15 minute run	3123 mwh	2678 mwh	445 mwh

There is another interesting result from the tests shown above. In the Process + Background column, it appears that the shorter Get tests could consume more power than the same Peek tests. This result appeared because the Get tests ran for much longer than anticipated. For example, the 3 minute tests ran over 4 minutes. This was because the application was asleep and it did not wake up to see that it was supposed to shut down until well after the time had elapsed. Thus, the length of time the application ran was much greater and the total power consumed was greater. To remove this anomaly, the overall tests were run for a longer period of time, as shown below:

Peek vs. Get Results #2 – Additional Results

Test Name	Test, Power Eval settings and duration	Ave Power Consumed Process + Background	Ave Power Consumed Background	Net Ave Power Consumed Process
M1 Get	GetMessage, Mode 1, 5 minute run	9.8 w	NA	NA
M2 Get	GetMessage, Mode 2, 5 minute run	10.1 w	10.1 w	0.0 w
M1 Get 15	GetMessage, Mode 1, 15 minute run	9.6 w	NA	NA
M2 Get 15	GetMessage, Mode 2, 15 minute run	9.6 w	9.6 w	0.1 w
M1 Peek	PeekMessage, Mode 1, 5 minute run	12.7 w	NA	NA
M2 Peek	PeekMessage, Mode 2, 5 minute run	11.3 w	9.9 w	1.5 w
M1 Peek 15	PeekMessage, Mode1, 15 minute run	12.3 w	NA	NA
M2 Peek 15	PeekMessage, Mode 2, 15 minute run	11.4 w	9.8 w	1.6 w

The nature of this data clears up the problem with the previous length of runs; here it's clear that the overall power consumed by the Peek tests is greater than that of the Get tests.

Peek vs. Get Results #3 – Additional Results

Test Name	Test, Power Eval settings and duration	Ave % total processor time Process + Background	Ave % total processor time Background	Processor Speed
M1 Get	GetMessage, Mode 1, 5 minute run	1%	NA	589 MHz
M2 Get	GetMessage, Mode 2, 5 minute run	1 %	1 %	589 MHz
M1 Get 15	GetMessage, Mode 1, 15 minute run	1 %	NA	589 MHz
M2 Get 15	GetMessage, Mode 2, 15 minute run	1 %	2 %	589 MHz
M1 Peek	PeekMessage, Mode 1, 5 minute run	99 %	NA	589 MHz
M2 Peek	PeekMessage, Mode 2, 5 minute run	99 %	0 %	589 MHz
M1 Peek 15	PeekMessage, Mode1, 15 minute run	100%	NA	589 MHz
M2 Peek 15	PeekMessage, Mode 2, 15 minute run	100 %	0 %	589 MHz

References and Notes

¹ KB93915 - MFC Application Idle Processing and Power Consumption

² KB40669 - Howto: Post Frequent Messages Within an Application

