Supporting Systems That Have More Than 64 Processors

Guidelines for Developers

November 5, 2008

Abstract

The 64-bit versions of Windows® 7 support more than 64 logical processors on a single machine. This paper provides information about the changes that some applications and drivers that run on Windows require to support this expanded number of processors.

This information applies for the Windows 7, 64-bit edition, operating system.

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

For the latest information, see:   
 http://www.microsoft.com/whdc/system/Sysinternals/MoreThan64proc.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.

© 2008 Microsoft Corporation. All rights reserved.

Microsoft, MSDN, 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 |  |  |  |
| November 5, 2008 | First publication | | | |

Contents

[Introduction 4](#_Toc213230599)

[Terminology 4](#_Toc213230600)

[Architectural Overview 5](#_Toc213230601)

[Group Creation 6](#_Toc213230602)

[Group Creation on NUMA Architectures 6](#_Toc213230603)

[Group Creation on Traditional Architectures 6](#_Toc213230604)

[Group, Process, and Thread Affinity 7](#_Toc213230605)

[System Thread Pool 8](#_Toc213230606)

[New and Changed Types and Macros 9](#_Toc213230607)

[KAFFINITY Type 9](#_Toc213230608)

[Group Number 9](#_Toc213230609)

[GROUP\_AFFINITY Structure 9](#_Toc213230610)

[MAXIMUM\_PROCESSORS Macro 9](#_Toc213230611)

[PROCESSOR\_NUMBER Structure 10](#_Toc213230612)

[Processor Index 10](#_Toc213230613)

[Application Modifications 10](#_Toc213230614)

[Setting Process Affinity 11](#_Toc213230615)

[Setting Thread Affinity and Ideal Processor 12](#_Toc213230616)

[New and Modified API Functions 13](#_Toc213230617)

[Kernel-Mode Driver Modifications 16](#_Toc213230618)

[Per-Processor Data Structures 16](#_Toc213230619)

[Static Array 16](#_Toc213230620)

[Dynamic Array 17](#_Toc213230621)

[Enumerating Processors 17](#_Toc213230622)

[Interrupt Affinity in Drivers 18](#_Toc213230623)

[Setting Group Interrupt Affinity Policy 18](#_Toc213230624)

[Changes to Resource Requirements and Resource Descriptors 19](#_Toc213230625)

[Changes to IoConnectInterruptEx 20](#_Toc213230626)

[Setting the Target Processor for DPCs 21](#_Toc213230627)

[New and Modified DDI Functions 21](#_Toc213230628)

[Resources 25](#_Toc213230629)

# Introduction

In the Windows® 7 operating system, the 64-bit kernel supports more than 64 logical processors. To scale up to support this expanded number of processors, some applications and Windows kernel-mode components require modification. This paper describes the terminology, architecture, and concepts that Windows 7 introduces to support more than 64 logical processors and provides information about the modifications that software might require.

For user-mode applications, the paper lists new Windows API functions and describes changes to existing functions and data structures that might be required to scale up.

For kernel-mode drivers, the paper lists the new driver device driver interfaces (DDIs) and describes changes to existing driver DDIs and data structures. In addition, it explains some techniques that the Microsoft kernel development team used to modify legacy drivers during prototype development.

# Terminology

Windows scale-up technology uses the following terms:

* **Logical processor**. One logical computing engine from the perspective of the operating system, application, or driver. In effect, a logical processor is a thread.
* **Core.** One processing unit, which can consist of one or more logical processors.
* **Processor**. One physical processor, which can consist of one or more cores. A physical processor is the same as a package, a socket, or a CPU.
* **Nonuniform memory architecture (NUMA) node**. A set of logical processors and cache that are close to one another.
* **Group**. A set of up to 64 logical processors.
* **Affinity**.A preference indicated by a thread, process, or interrupt for operation on a particular processor, node, or group.

Figure 1 shows the relationships among the objects that these terms represent.



Figure 1. Scale-up terminology

# Architectural Overview

Windows support for more than 64 logical processors is based on a new concept—the *group.* A group is a static set of up to 64 logical processors that is treated as a single scheduling entity. Groups have the following characteristics:

* The Windows kernel determines at boot time which processor belongs to which group.
* Each logical processor is assigned to a single group.
* All the logical processors in a core, and all the cores in a physical processor, are assigned to the same group if possible.
* Physical processors that are physically close to one another are assigned to the same group.
* A process can have affinity for more than one group at a time. However, a thread can be assigned to only a single group at any time, and that group is always in the affinity of the thread’s process.
* An interrupt can only target processors of a single group.
* In NUMA architectures, a group can contain processors from one or more nodes, but all the processors in a node are assigned to the same group whenever possible.

The group architecture assumes that related code runs on the processors in the same group and that best performance results if the processors in a group are physically close to one another. This architecture has several benefits:

* Many existing drivers and applications can run without modification on systems that have fewer than 64 logical processors.
* Groups provide locality of hardware that is used by related software components, thus avoiding an adverse effect on performance.
* Software can determine the relationships among processors and groups by using exposed interfaces.
* The group architecture is easily extensible to support additional processors in the future.

Figure 2 shows a hypothetical system that has multiple processor groups.

Figure 2. Processor groups

The system in Figure 2 has four processor groups, which is the maximum that is supported in Windows 7. Group 0 contains 2 NUMA nodes that have 32 logical processors each. Groups 1, 2, and 3 each contain a single NUMA node that has 64 logical processors.

# Group Creation

At startup, Windows 7 creates processor groups according to two basic principles:

* Minimize the number of groups in a system.
* Maximize locality between the processors in a group.

The first principle ensures that systems with fewer than 64 logical processors always have a single group. Most current hardware falls into this category, so most systems have a single group. The existing API and DDI functions continue to work exactly like on earlier versions of Windows, so that existing applications and drivers that are not targeted at very large systems can run without modification.

The second principle provides better performance because of better cache utilization among the threads of a particular process. Entire NUMA nodes are assigned to the same group, so that a node is a subset of a group. If multiple nodes are assigned to a single group, Windows chooses nodes that are physically close to one another.

Minimizing both the number of groups and the distance between nodes in a group is not always possible, but Windows balances these factors when it forms groups.

In Windows 7, administrators cannot control group formation. However, the system exposes interfaces so that both kernel-mode drivers and user-mode applications can have full information about the number and contents of the groups.

## Group Creation on NUMA Architectures

On NUMA architectures, Windows uses the capacity of the NUMA nodes to determine group assignments. The capacity of a node is defined as the number of processors that are present at boot time together with any additional logical processors that can be hot-added dynamically.

By default, an entire node is assigned to a group. However, if the capacity of a node is greater than the maximum number of logical processors in a group (64), the system splits the node into *n* groups, where the first *n*-1 groups have capacities that are the same as the group size.

If the capacities of the NUMA nodes are fairly small, the system can assign more than one node to the same group. To maximize hardware locality, the system uses the distances between the nodes to determine which nodes should be grouped together.

## Group Creation on Traditional Architectures

On traditional, non-NUMA architectures, Windows similarly considers the number of logical processors that are present at boot together with any logical processors that can be hot-added later.

If the total number of logical processors is less than or equal to the maximum group size (currently 64), Windows assigns all the logical processors to group 0. If the number of logical processors exceeds the maximum group size, Windows creates multiple groups by splitting the node into *n* groups, where the first *n*-1 groups have capacities that are equal to the group size.

# Group, Process, and Thread Affinity

In earlier versions of Windows, a process or thread could specify an affinity for a particular processor, so that the thread or process was guaranteed to run on that processor. Windows 7 expands this notion of affinity to apply to groups and to the processors in a group.

Windows 7 uses the following defaults for affinity:

* Windows 7 initially assigns each process to a single group in a round-robin manner across the groups in the system. A process starts its execution assigned to exactly one group.
* The first thread of a process initially runs in the group to which Windows assigns the process. However, an application can override this default as described in “Setting Process Affinity” later in this paper.
* Each newly created thread is by default assigned to the same group as the thread that created it. However, at thread creation, an application can specify the group to which the thread is assigned.
* Only the system process is assigned a multigroup affinity at startup time. All other processes must explicitly assign threads to a different group to use the full set of processors in the system.

Over time, a process can expand to contain threads that are running on all groups in a machine, but a single thread can never be assigned to more than one group at any time. However, a thread can change the group to which it is assigned.

The reason for initially limiting all threads to a single group is that 64 processors is more than adequate for the typical application. An application that requires the use of multiple groups so that it can run on more than 64 processors must intentionally determine where to run its threads. The application is responsible for setting thread affinities to the desired groups.

The effect of the defaults is to constrain applications to a single group unless they explicitly create threads on other groups. In a group, most traditional uses of processor affinity operate exactly like on earlier versions of Windows. That is, applications that constrain their operation to a single group do not require modification to operate correctly on a machine that has more than 64 logical processors. Most applications benefit by this greater locality of resources. Furthermore, unless an application explicitly changes a process affinity mask or assigns work to a different group, the application can run with a group-relative view of the system.

Both drivers and applications can change the defaults by modifying thread and process affinity. In addition, drivers can set an application thread’s temporary “system affinity,” which later reverts to its previous setting. The temporary affinity does not permanently change the affinity of the thread.

To run a piece of work in a different group, a process must explicitly assign the work to that group. To scale an application efficiently across multiple groups, you must understand what pieces of work can run essentially independently from other pieces. The group architecture assumes that the software developer has detailed knowledge of the characteristics of the application’s workload and thus is better suited to make these explicit choices than is the operating system.

Although a large application might scale more efficiently by breaking its workload into sections and assigning unrelated sections to threads that are running in different groups, this is not always true in NUMA architectures. In NUMA architectures, this approach can result in the execution of unrelated work on physically distant processors, because all the processors in a NUMA node typically belong to the same group. Such results can actually hinder performance.

Therefore, applications that scale beyond 64 logical processors should organize their work distribution schemes around the NUMA node concept. A process can set a preferred NUMA node, which indicates to the Windows scheduler that the process should run on the processors in that node if possible. A thread can do the same. In both cases, the application can set this preference when it creates the process or thread, so that initial memory allocations occur on the preferred node for optimal performance.

# System Thread Pool

In Windows 7, the system thread pool is extended to provide per-node queuing. When a thread queues a work item, Windows runs the work item in a thread that is assigned to the same node as the queuing thread if possible. If a thread in the same node is not available, Windows guarantees that the work item runs in the same group from which it was queued. Therefore, a component that is constrained to a particular group always runs thread pool work items in that group. A component that does not operate correctly in a multigroup system can use the thread pool without problems.

This queuing mechanism helps preserve execution locality and improve performance, because work items usually share data with the queuing thread.

However, if a change in the affinity of the worker thread might have undesirable side-effects, Windows does not guarantee that the work item runs on a particular node or group. Specifically, node and group guarantees do not apply if any of the following are true:

* The work item is flagged as persistent by a call to the new **SetThreadpoolCallbackPersistent** function.
* The work item runs on a waiter thread (WT\_EXECUTEINWAITTHREAD).
* The work item runs on a timer thread (WT\_EXECUTEINTIMERTHREAD).
* The work item runs on a persistent I/O thread (WT\_EXECUTEINPERSISTENTIOTHREAD or WT\_EXECUTEINPERSISTENTTHREAD).

# New and Changed Types and Macros

Windows 7 takes a group-oriented view toward processor affinities and processor numbers and defines new structures for this purpose. This section describes the basic data types, structures, and macros that are used to identify processor numbers, groups, and affinities:

* KAFFINITY
* Group number
* GROUP\_AFFINITY
* MAXIMUM\_PROCESSORS
* PROCESSOR\_NUMBER
* Processor index

## KAFFINITY Type

In earlier versions of Windows, processor affinity is represented by a single 64-bit integer, which is defined as KAFFINITY. When the number of logical processors can exceed 64, a single 64-bit integer is no longer sufficient to represent the processor affinity. Therefore, for Windows 7, all existing structures and APIs that use KAFFINITY must either be extended or will change behavior.

For an application that is assigned to a single group, the group number is implied. Therefore, the traditional KAFFINITY structure and UCHAR processor number are still valid concepts for describing group-relative processors and processor sets.

## Group Number

A group is represented by a group number, which is a USHORT. Windows 7 supports a maximum of four groups, numbered from 0 to 3.

## GROUP\_AFFINITY Structure

The GROUP\_AFFINITY structure describes a group-relative processor affinity. It is defined as follows:

typedef struct \_GROUP\_AFFINITY {

KAFFINITY Mask;

USHORT Group;

USHORT Reserved[3];

} GROUP\_AFFINITY, \*PGROUP\_AFFINITY;

The **Mask** member of the structure is a bitmask that specifies affinity for zero or more processors within the group. The **Group** member contains the group number.

## MAXIMUM\_PROCESSORS Macro

In Windows 7, the MAXIMUM\_PROCESSORS macro does not mean the maximum number of processors that the system supports. It is kept for backward compatibility for user-level applications, but it now means the maximum number of logical processors in a group. For 64-bit Windows, the following macros are used to define the maximum number of groups and processors that the system supports:

#define MAXIMUM\_PROC\_PER\_GROUP 64

#define MAXIMUM\_PROCESSORS MAXIMUM\_PROC\_PER\_GROUP

To provide for future increases in the number of supported processors and groups, Windows does not expose statically-defined macros that indicate the maximum number of groups and the maximum number of logical processors. Instead, these values are exposed to applications by the new **GetMaximumProcessorGroupCount** and **GetMaximumProcessorCount** functions, and to kernel-mode drivers by the **KeQueryMaximumGroupCount** and **KeQueryMaximumProcessorCountEx** functions, respectively.

## PROCESSOR\_NUMBER Structure

The PROCESSOR\_NUMBER structure describes a processor and the group to which it belongs, as follows:

typedef struct \_PROCESSOR\_NUMBER {

USHORT Group;

UCHAR Number;

UCHAR Reserved;

} PROCESSOR\_NUMBER, \*PPROCESSOR\_NUMBER;

The **Group** member of the structure contains the group number, and the **Number** member contains the zero-based processor number. When the system contains only one group, the **Number** member is the same as the UCHAR processor number in earlier versions of Windows.

For each group, the affinity mask of active logical processors in the group is contiguous from bit 0 through bit *n*–1, where *n* is the number of active logical processors in the group.

## Processor Index

Kernel-mode components can access a system-wide processor number. In earlier versions of Windows, a processor number was represented by a ULONG. This number is now called the processor index and is still represented by a ULONG. It is not exposed to applications.

The processor index is a system-wide integer that uniquely identifies a logical processor. The processor index is minimally exposed to device drivers and is primarily used as an index to arrays of per-processor data structures that the driver maintains. The **KeGetCurrentProcessorNumberEx** function returns the processor index and a PROCESSOR\_NUMBER structure that contains the processor number.

Driver developers can safely assume that processor indexes are sequential and continuous, starting from zero to the number of active logical processors in the system minus one.

# Application Modifications

To operate correctly on systems with more than 64 logical processors, some applications require modification to accommodate the additional data that is required to describe the processors:

* Applications that manage, maintain, or display per-processor information for the entire system must be modified to support more than 64 logical processors. An example of such a tool is Windows Task Manager, which displays the workload of each processor in the system.
* Applications that are performance critical and can scale efficiently beyond 64 logical processors must be modified to run on such systems. For example, database applications might benefit from such modifications.
* If an application uses a DLL that has per-processor data structures and has not been modified to support more than 64 processors, all threads in the application that call functions that are exported by the DLL must be assigned to the same group.

Applications that do not call any API functions that use processor affinity masks or processor numbers continue to operate correctly on all systems. More important, all existing applications continue to operate correctly in a Windows 7 system that has 64 or fewer logical processors.

The following sections describe modifications that applications might require to scale to support additional processors.

## Setting Process Affinity

As previously mentioned, each process is by default assigned to exactly one group, which Windows chooses in a round-robin manner. An application can override this default in the following ways:

* By passing the INHERIT\_PARENT\_AFFINITY flag to **CreateProcess**.

This flag causes Windows to use the affinity of the parent process to generate the affinity for the new process. If the parent process contains threads in multiple groups, Windows arbitrarily selects the initial group from among those groups. This flag is useful for keeping a tree of related processes assigned to close processors.

The INHERIT\_PARENT\_AFFINITY flag remains in effect for the children and later descendants of the created process until a process explicitly specifies a different preference. That is, if a process in the lineage of the parent was started with INHERIT\_PARENT\_AFFINITY, the new process also inherits the parent’s affinity in the same manner.

* By passing the PROC\_THREAD\_ATTRIBUTE\_PREFERRED\_NODE extended attribute to **CreateProcess**.

This attribute specifies a preferred NUMA node for the new process and causes Windows to assign the new process to the group that contains the specified node, assuming that it is a valid node and contains processors. The process affinity is set to all logical processors in the node’s group, including any processors that belong to other nodes.

To determine the processor on which to run a thread, the Windows scheduler first examines the thread’s preferred node. If no processor is available on the preferred node, the scheduler searches the thread’s entire affinity mask.

Note that this is a “preferred” node that is provided as a performance optimization and is not guaranteed to be honored. The process might not run on the preferred node, and it might not be assigned to the same group as the preferred node. Other factors, such as resource management limitations on the process, could override the specified preference.

If a process is assigned to a single group, an application can modify the process affinity mask to include an arbitrary set of logical processors in that group. These changes are reflected to all threads in the process. This behavior has the same effect as changing the process affinity mask on Windows Server® 2008 and earlier versions of Windows, which support only 64 logical processors (one group).

If a process extends beyond a group, however, the concept of process affinity as defined before Windows 7 becomes obsolete. The threads in the process can change their affinity to run in a different group. Therefore, process affinity becomes the union of all thread affinities.

## Setting Thread Affinity and Ideal Processor

Threads can run only on the logical processors in a single group. By default, the thread affinity is all logical processors in the parent thread’s group. Windows assigns threads across logical processors within the thread’s affinity mask according to thread priority. At thread creation, an application can change the default thread affinity and can specify an ideal processor for a thread by calling the new **CreateRemoteThreadEx** function.

The ideal processor is the logical processor on which the Windows scheduler tries to run the thread whenever possible. The scheduler searches for a processor in the following order:

1. The thread’s ideal processor.

2. A processor in the thread’s preferred NUMA node.

3. Other processors in the thread affinity mask.

To specify the group affinity for a thread at creation

* Call **CreateRemoteThreadEx** and pass the PROC\_THREAD\_ATTRIBUTE  
  \_GROUP\_AFFINITY extended attribute together with a GROUP\_AFFINITY structure.

For more information about how to specify extended attributes for threads, see the **InitializeProcThreadAttributeList** function on MSDN®, which is listed in “Resources.”

To change the affinity of an existing thread

* Call either the existing **SetThreadAffinityMask** function or the new **SetThreadGroupAffinity** function.

**SetThreadAffinityMask** modifies the affinity of the thread without affecting its group assignment. This function takes a group-relative affinity—which is the same as the thread affinity mask that is used in Windows Server 2008 and earlier versions.

**SetThreadGroupAffinity** takes a GROUP\_AFFINITY structure and can be used to change the group-relative affinity, the group assignment, or both.

To specify the ideal processor at thread creation

* Pass the PROC\_THREAD\_ATTRIBUTE\_IDEAL\_PROCESSOR extended attribute to **CreateRemoteThreadEx** together with a PROCESSOR\_NUMBER structure.

An application can specify both an ideal processor and a group affinity in a single call to **CreateRemoteThreadEx**. If so, the group affinity must contain the ideal processor. Otherwise, Windows assigns a group affinity for the thread that comprises the ideal processor’s group and the active processors that are currently in the group. To improve performance, Windows allocates the thread’s user-mode stack on the same NUMA node as the specified ideal processor.

## New and Modified API Functions

Extending Windows to support more than 64 processors requires several new API functions in addition to changes to some existing functions. Applications that call the existing functions must ensure that the intended use corresponds to the new behavior. In some cases, applications must replace calls to the existing functions with calls to new functions.

Generally, existing functions that previously returned information about all the processors on the system now return the same information for processors in group 0. New functions return group-specific or system-wide information.

In Windows 7 and later versions, a group-relative affinity mask of zero is valid in calls to functions that require such a mask. A zero affinity mask indicates that the thread can run on all the logical processors on which the process can run within the specified group. If the mask is nonzero, it must be a valid subset of the currently active logical processors in the specified group. If the mask contains an invalid logical processor, the function fails.

In Windows Server 2008 and earlier versions, a zero affinity mask was not valid. In addition, if the caller specified a nonexistent processor in the affinity mask, Windows removed the invalid processor and the function completed successfully.

Table 1 lists the new and modified user-mode API functions that support additional processors in Windows 7. For detailed descriptions of the existing functions, see [the](http://baseteam/coreplatform/scaleup/Shared%20Documents/Greater%20Than%2064LP%20Design%20Spec.docx) “Win32 and COM Development” documentation on MSDN, which is listed in “Resources.” Complete documentation for the new functions will appear in the Windows 7 Software Development Kit (SDK).

Table 1. New and Modified API Functions for More Than 64 Processors

|  |  |  |
| --- | --- | --- |
| Function name | New or changed? | Description |
| **CreateRemoteThreadEx** | New | Enables an application to change the default thread group affinity and specify an ideal processor for a thread. |
| **GetActiveProcessorCount** | New | Returns the number of active logical processors in a group or in the system. |
| **GetActiveProcessorGroupCount** | New | Returns the number of active groups in the system. |
| **GetCurrentProcessorNumber** | Changed | Returns the group-relative number of the logical processor (in DWORD) on which the calling thread is running. |
| **GetCurrentProcessorNumberEx** | New | Returns a PROCESOR\_NUMBER structure that indicates the logical processor on which the calling thread is running. |
| **GetLogicalProcessorInformation** | Changed | Returns the logical processor information (package, cache, NUMA, and so on) of processors in the group to which the calling thread is currently assigned. |
| **GetLogicalProcessorInformationEx** | New | Returns the logical processor information for all processors in the entire system. In addition, the LOGICAL\_PROCESSOR\_RELATIONSHIP enumeration defines a new constant, **RelationGroup**, which indicates the number of groups in the system and the number of processors in each group. |
| **GetMaximumProcessorCount** | New | Returns the maximum number of logical processors that a group or the system can support. |
| **GetMaximumProcessorGroupCount** | New | Returns the maximum number of groups that the system supports. |
| **GetNumaAvailableMemoryNode** | Changed | Returns the amount of memory that is available in a specified node (in UCHAR). |
| **GetNumaAvailableMemoryNodeEx** | New | Returns the amount of memory that is available in a specified node (in USHORT). The only difference is the data type of the node. |
| **GetNumaNodeProcessorMask** | Changed | Returns the affinity mask (in ULONGLONG) of all logical processors for the specified node (in UCHAR) if the node belongs to the same group as the calling thread. Otherwise, it returns 0 as the affinity mask. |
| **GetNumaNodeProcessorMaskEx** | New | Returns the group affinity mask (in GROUP\_AFFINITY) of all logical processors in the specified node (in USHORT) regardless of which group the node belongs to. |
| **GetNumaProcessorNode** | Changed | Returns the node number (in UCHAR) of a specified logical processor (in UCHAR) if the node belongs to the same group as the calling thread. The processor number is the group-relative number. |
| **GetNumaProcessorNodeEx** | New | Returns the node number (in USHORT) of a given logical processor (in PROCESSOR\_NUMBER). The node can belong to any group in the system. |
| **GetNumaProximityNode** | Changed | Returns the node number (in UCHAR) for a specified proximity identifier. |
| **GetNumaProximityNodeEx** | New | Returns the node number (in USHORT) for a specified proximity identifier. The only difference between this function and **GetNumaProximityNode** is the data type of the node. |
| **GetProcessAffinityMask** | Changed | Returns the group-relative process affinity (a DWORD\_PTR) and the group-relative processor mask (a DWORD\_PTR) of the active logical processors if the calling process contains threads in a single group.  If the calling process contains threads in multiple groups, the function returns 0 for both affinity masks. |
| **GetProcessGroupAffinity** | New | Returns the current group affinity of the process (in GROUP\_AFFINITY). |
| **GetSystemInfo** | Changed | Returns the number and affinity mask of active logical processors in the current thread’s group in the **NumberOfProcessors** and **ActiveProcessorsAffinityMask** members of the SYSTEM\_BASIC\_INFORMATION structure, respectively. |
| **GetThreadGroupAffinity** | New | Returns the current group affinity of the thread (in GROUP\_AFFINITY). |
| **QueryIdleProcessorCycleTime** | Changed | Returns the cycle time for the idle threads of all logical processors in the group to which the calling thread is assigned. |
| **QueryIdleProcessorCycleTimeEx** | New | Returns the cycle time for the idle threads of all logical processors in the specified group. |
| **SetProcessAffinityMask** | Changed | Sets the affinity of the process to the specified group-relative affinity mask (a DWORD\_PTR) if the calling process is assigned to a single group. This function does not change the group assignment of the process.  If the calling process contains threads in multiple groups, the function returns the ERROR\_INVALID\_PARAMETER error status. |
| **SetThreadAffinityMask** | Changed | Sets affinity of the thread (as a DWORD\_PTR) to one or more logical processors within the thread’s current group. |
| **SetThreadGroupAffinity** | New | Sets the affinity of the thread to a set of logical processors in a specified group (in GROUP\_AFFINITY). |
| **SetThreadIdealProcessor** | Changed | Sets the ideal processor for the thread to a logical processor in the thread’s current group. |

# Kernel-Mode Driver Modifications

To operate correctly on systems with more than 64 logical processors, some kernel-mode drivers must be modified to accommodate the additional data that is required to describe the processors. A driver that does any of the following requires modification:

* Calls any device driver interface (DDI) that uses processor affinity masks or processor numbers.
* Is performance critical and affinitizes interrupt and deferred procedure call (DPC) workload beyond the first 64 processors.

Drivers that do not call any DDIs that use processor affinity masks or processor numbers still operate correctly. Furthermore, all existing unmodified legacy drivers continue to operate correctly on a Windows 7 system that has 64 or fewer logical processors.

Drivers that use the new driver interfaces that are described in this paper also run correctly on older versions of Windows. Microsoft provides a version compatibility library that converts the declarations of the new interfaces to legacy versions on older systems.

The following sections describe modifications that might be required to scale drivers to run on systems that have more than 64 logical processors.

## Per-Processor Data Structures

Drivers that use per-processor data structures typically retrieve a processor number and then store information about each processor in an array.

To get the processor number, existing drivers call **KeGetCurrentProcessorNumber**. In Windows 7, this function returns the group-relative processor number if the caller is running on a logical processor in group 0. Therefore, drivers that run only on systems that have fewer than MAXIMUM\_PROC\_PER\_GROUP logical processors do not require any changes.

Drivers that run on systems that can have more than one group should instead use the new **KeGetCurrentProcessorNumberEx** function. This function returns both the processor index (in ULONG) and the processor number (in PROCESSOR\_NUMBER) of the logical processor on which the caller is running.

Most existing drivers use a static array of MAXIMUM\_PROCESSORS size for each per-processor data structure. Alternatively, some drivers use a dynamic array and call **KeQueryMaximumProcessorCount** to get the array size. The following sections describe common ways to modify a per-processor data structure.

### Static Array

If the driver has access to the definition of the MAXIMUM\_PROC\_PER\_SYSTEM macro, the driver can use a static array of MAXIMUM\_PROC\_PER\_SYSTEM size regardless of whether the per-processor data structure is a stand-alone data array or is a member of a bigger structure.

### Dynamic Array

If the driver uses a dynamic array or does not have access to the definition of the MAXIMUM\_PROC\_PER\_SYSTEM macro, the driver must obtain the required size for the array and dynamically allocate memory for the array. To determine the number of elements that are required in the array, the driver can call **KeQueryMaximumProcessorCountEx** (ALL\_PROCESSOR\_GROUPS).

If the array is a stand-alone data structure, the driver can dynamically allocate the memory in the usual manner. If the array is part of larger structure, the driver has three options:

* Declare the per-processor data structure as a pointer and dynamically allocate the memory for the array.
* Declare the per-processor data structure as a pointer and let it point just beyond the last member of the larger structure. Dynamically allocate memory for the larger structure. The amount of memory to allocate for the bigger structure is the size of the structure plus the actual size of the array.
* Declare the per-processor data structure as an array of size 1 and make it the last member of the larger structure. Dynamically allocate memory for the larger structure. The amount of memory to allocate for the larger structure is the size of the structure up to (but excluding) the array plus the actual size of the array.

If the larger structure contains two or more per-processor data arrays, use either of the first two options.

## Enumerating Processors

Drivers that enumerate processors require changes to support groups.

The following code sample enumerates all logical processors in a group affinity structure:

GROUP\_AFFINITY Affinity; // input

KAFFINITY Mask;

PROCESSOR\_NUMBER ProcNumber;

ProcNumber.Group = Affinity.Group;

ProcNumber.Number = 0;

ProcNumber.Reserved = 0;

Mask = Affinity.Mask;

while (Mask != 0) {

if ((Mask & 1) != 0) {

// ProcNumber is an enumerated processor.

}

Mask >>= 1;

ProcNumber.Number += 1;

}

The following code sample enumerates all active logical processors in a group:

USHORT Group; // input

UCHAR Count;

UCHAR GroupIndex;

PROCESSOR\_NUMBER ProcNumber;

Count = (UCHAR)KeQueryActiveProcessorCountEx(Group);

ProcNumber.Group = Group;

ProcNumber.Reserved = 0;

for (GroupIndex = 0; GroupIndex < Count; GroupIndex += 1) {

ProcNumber.Number = GroupIndex;

// ProcNumber is an enumerated processor.

}

The following code sample enumerates all active logical processors in the system:

ULONG Count;

ULONG ProcIndex;

PROCESSOR\_NUMBER ProcNumber;

Count = KeQueryActiveProcessorCountEx(ALL\_PROCESSOR\_GROUPS);

for (ProcIndex = 0; ProcIndex < Count; ProcIndex += 1) {

KeGetProcessorNumberFromIndex(ProcIndex, &ProcNumber);

// ProcNumber is an enumerated processor.

}

## Interrupt Affinity in Drivers

By default, Windows 7 assigns interrupt vectors only to logical processors in group 0. A driver can set a different interrupt affinity for its device in code or by setting a registry key. However, all the logical processors that are targets of a particular interrupt must be in a single group.

This section describes how a driver can set an affinity policy for its device, and covers changes to the resource-descriptor data structures and the **IoConnectInterruptEx** function to support interrupt affinity across processor groups.

The DPC for an interrupt always runs on the same logical processor that was interrupted unless the driver specifies otherwise, as described in “Setting the Target Processor for DPCs” later in this paper.

### Setting Group Interrupt Affinity Policy

A driver can specify the group affinity policy of the interrupts that its device can generate either in its INF file or when it handles an IRP\_MJ\_PNP request with the IRP\_MN\_FILTER\_RESOURCE\_REQUIREMENTS minor code.

By default, all devices interrupt on group 0. To enable a device to interrupt on logical processors that are not in group 0, a driver must either:

* Specify 1 as the Group Policy valuein the INF file.

—or—

* Specify the group number for each interrupt in the driver’s response to the IRP\_MN\_FILTER\_RESOURCE\_REQUIREMENTS request, as described in the next section.

If the driver sets group affinity policy, it must also define the NT\_PROCESSOR\_GROUPS macro by doing one of the following:

* Including the ntosp.h header file.
* Defining NT\_PROCESSOR\_GROUPS in its source code before including ntddk.h or wdm.h.
* Defining NT\_PROCESSOR\_GROUPS in its *Sources* file.

To set group affinity policy in the driver’s INF file

* Specify a value in the **AddReg** directive as follows:

[HKR, "Interrupt Management\Affinity Policy", "GroupPolicy", group-value]

where *group-value* is one of the following:

**0** indicates that the device interrupts only on logical processors in group 0.

**1** indicates that the device can interrupt on logical processors beyond group 0.

The following section explains how a driver sets the group affinity policy when it responds to the IRP\_MJ\_PNP/IRP\_MN\_FILTER\_RESOURCE\_REQUIREMENTS request.

### Changes to Resource Requirements and Resource Descriptors

In Windows 7, the IO\_RESOURCE\_DESCRIPTOR and CM\_PARTIAL\_RESOURCE\_DESCRIPTOR structures can have an additional member, **Group**, of type USHORT. In addition, the **Flags** member of the IO\_RESOURCE\_DESCRIPTOR structure supports a new value: CM\_RESOURCE\_INTERRUPT\_POLICY\_INCLUDED.

These changes enable drivers to request that their device interrupt on a particular group.

In Windows 7, IO\_RESOURCE\_DESCRIPTOR is defined as follows:

typedef struct \_IO\_RESOURCE\_DESCRIPTOR {

…//Additional fields omitted

struct {

ULONG MinimumVector;

ULONG MaximumVector;

#if defined(NT\_PROCESSOR\_GROUPS)

IRQ\_DEVICE\_POLICY AffinityPolicy;

USHORT Group;

#else

IRQ\_DEVICE\_POLICY AffinityPolicy;

#endif

IRQ\_PRIORITY PriorityPolicy;

KAFFINITY TargetedProcessors;

} Interrupt;

…//Additional fields omitted

}

CM\_PARTIAL\_RESOURCE\_DESCRIPTOR is defined as follows:

typedef struct \_CM\_PARTIAL\_RESOURCE\_DESCRIPTOR {

…//Additional fields omitted

struct {

#if defined(NT\_PROCESSOR\_GROUPS)

USHORT Level;

USHORT Group;

#else

ULONG Level;

#endif

ULONG Vector;

KAFFINITY Affinity;

} Interrupt;

…//Additional fields omitted

}

If the NT\_PROCESSOR\_GROUPS macro is defined, the **Group** member is added to both structures to indicate the requested and assigned group number for the target processors of an interrupt. For IO\_RESOURCE\_DESCRIPTOR, this number can be ALL\_PROCESSOR\_GROUPS (0xffff) to indicate any group. The sizes of both structures remain unchanged.

To set the group affinity of an interrupt while handling the IRP\_MN\_FILTER\_RESOURCE\_REQUIREMENTS request

1. Set the new CM\_RESOURCE\_INTERRUPT\_POLICY\_INCLUDED value in the **Flags** member of the IO\_RESOURCE\_DESCRIPTORstructure for the interrupt. This setting indicates that the resource descriptor structure contains the group affinity.

2. Specify the group affinity in one of the following ways:

To specify affinity for a particular group, set the **AffinityPolicy** member of the IO\_RESOURCE\_DESCRIPTORstructure to **IrqPolicySpecifiedProcessors** and set the **Group** member of the IO\_RESOURCE\_DESCRIPTORstructure to the appropriate group number. The driver must also specify the target processors within the group in the **TargetedProcessors** member.

To specify affinity for any group, set the **Group** member of the IO\_RESOURCE\_DESCRIPTORstructure to ALL\_PROCESSOR\_GROUPS and the **AffinityPolicy** member to any value except **IrqPolicySpecifiedProcessors**. A driver cannot specify target processors if it uses ALL\_PROCESSOR\_GROUPS; such target specifications are ignored.

### Changes to IoConnectInterruptEx

In Windows 7, the IO\_CONNECT\_INTERRUPT\_PARAMETERS structure that is passed to **IoConnectInterruptEx** changes to support processor groups.

The **Version** member of the IO\_CONNECT\_INTERRUPT\_PARAMETERS structure has a new possible value: CONNECT\_FULLY\_SPECIFIED\_GROUP.

The IO\_CONNECT\_INTERRUPT\_FULLY\_SPECIFIED\_PARAMETERS structure has a new member: **Group** (of type USHORT).

To fully specify the interrupt connection for an interrupt that is targeted at processors beyond group 0, a driver must supply CONNECT\_FULLY\_SPECIFIED\_GROUP as the version number and supply a value in the **Group** member of the IO\_CONNECT\_INTERRUPT\_FULLY\_SPECIFIED\_PARAMETERS structure.

## Setting the Target Processor for DPCs

Legacy drivers call **KeSetTargetProcessorDpc** to specify a logical processor as the target for a DPC. This function takes a UCHAR processor index as a parameter and lets the target of a DPC be a logical processor that is not currently present but might be hot-added later.

New drivers call **KeSetTargetProcessorDpcEx** to set the target processor for a DPC. Unlike the **KeSetTargetProcessorDpc** function, **KeSetTargetProcessorDpcEx** specifies a logical processor number (in PROCESSOR\_NUMBER) as the DPC target and requires that the processor be present.

Consider the following legacy code:

KDPC Dpc[MAXIMUM\_PROCESSORS];

UCHAR Processor;

for (Processor = 0; Processor < MAXIMUM\_PROCESSORS; Processor += 1) {

KeInitializeDpc(Dpc + Processor, …);

KeSetTargetProcessorDpc(Processor);

}

This code adds all possible logical processors as DPC targets, whether or not they are currently present on the machine.

The following code was modified to support more than 64 logical processors but is not functionally the same as the original code because the system might not have that many processors. For any logical processor that is not present, **KeGetProcessorNumberFromIndex** returns STATUS\_INVALID\_PARAMETER.

KDPC Dpc[MAXIMUM\_PROC\_PER\_SYSTEM];

ULONG ProcIndex;

PROCESSOR\_NUMBER ProcNumber;

for (ProcIndex = 0;   
 ProcIndex < MAXIMUM\_PROC\_PER\_SYSTEM; ProcIndex += 1) {

KeInitializeDpc(Dpc + ProcIndex, …);

if (KeGetProcessorNumberFromIndex(ProcIndex, &ProcNumber) ==

STATUS\_SUCCESS) {

KeSetTargetProcessorDpcEx(&ProcNumber);

}

}

**KeSetTargetProcessorDpcEx** cannot set a logical processor as a DPC target until that logical processor is present. When a new logical processor is hot-added to the system, the driver’s hot-add notification callback function should call **KeSetTargetProcessorDpcEx** to add the logical processor as a DPC target.

## New and Modified DDI Functions

The behavior of several existing driver DDIs changes to support more than 64 logical processors. Any driver that calls these functions must ensure that the intended use corresponds to the new behavior. In some cases, drivers must replace calls to the existing functions with calls to new functions that support additional processors.

In general, existing functions that previously returned information about all the processors on the system now return the same information for logical processors in group 0. New functions return group-specific or system-wide information.

Table 2 lists the new and changed DDI functions. For detailed descriptions of the existing DDIs, see [the](http://baseteam/coreplatform/scaleup/Shared%20Documents/Greater%20Than%2064LP%20Design%20Spec.docx) Windows Driver Kit (WDK). Complete documentation for the new DDIs will appear in the Windows 7 WDK.

Table 2. New and Changed Kernel-Mode Functions to Support More Than 64 Processors

|  |  |  |
| --- | --- | --- |
| Function name | New or changed? | Description |
| **IoGetAffinityInterrupt** | New | Returns the assigned group affinity (in GROUP\_AFFINITY) of an interrupt, given the returned interrupt object after successfully connecting an interrupt by **IoConnectInterruptEx** or **IoConnectInterrupt**. |
| **IoGetDeviceNumaNode** | New | Returns the node number of the specified device, if known. |
| **KeGetCurrentNodeNumber** | New | Returns the node number for the current processor. |
| **KeGetCurrentProcessorNumber** | Changed | Returns the group-relative processor number if the caller is running on a logical processor in group 0. If the caller is running on a logical processor in any other group, it returns an integer that is less than the number of active logical processors in group 0. |
| **KeGetCurrentProcessorNumberEx** | New | Returns both the processor index (in ULONG) and the processor number (in PROCESSOR\_NUMBER) of the logical processor on which the caller is running. |
| **KeGetProcessorIndexFromNumber** | New | Returns the corresponding processor index (in ULONG) from a logical process number (in PROCESSOR\_NUMBER). |
| **KeGetProcessorNumberFromIndex** | New | Returns the corresponding logical processor number (in PROCESSOR\_NUMBER) from a processor index (in ULONG). |
| **KeQueryActiveGroupCount** | New | Returns the number of active groups in the system. |
| **KeQueryActiveProcessorCount** | Changed | Returns the number and affinity mask of active logical processors in group 0. |
| **KeQueryActiveProcessorCountEx** | New | Returns the number of active logical processors in a given group or in the entire system. |
| KeQueryActiveProcessors | Changed | Returns the affinity mask of active logical processors in group 0. |
| **KeQueryGroupAffinity** | New | Returns the affinity mask of active logical processors in a given group. |
| **KeQueryHighestNodeNumber** | New | Returns the highest possible node number on the system. |
| **KeQueryLogicalProcessorRelationship** | New | Returns a specified type of relationship information about groups, physical processors, and nodes in the host system. |
| **KeQueryMaximumGroupCount** | New | Returns the maximum number of groups that the system supports. |
| **KeQueryMaximumProcessorCount** | Changed | Returns the maximum number of logical processors that can be in group 0. |
| **KeQueryMaximumProcessorCountEx** | New | Returns the maximum number of logical processors that can be in a given group or in the entire system. The maximum number of logical processors in a group is less than or equal to MAXIMUM\_PROC\_PER\_GROUP. |
| **KeQueryNodeActiveAffinity** | New | Returns the current processor affinity of a given NUMA node. |
| **KeQueryNodeMaximumProcessorCount** | New | Returns the maximum number of logical processors that can appear in the given NUMA node, including dynamic processors. |
| **KeRevertToUserAffinityThreadEx** | Changed | Sets the system group affinity of the current thread to group 0 and sets the group-relative affinity (in KAFFINITY) to a specified nonzero value. If the specified group-relative affinity is zero, **KeRevertToUserAffinityThreadEx** restores the user group affinity of the thread to its user group affinity. |
| **KeRevertToUserGroupAffinityThread** | New | Sets the system group affinity of the current thread to the specified group affinity (in GROUP\_AFFINITY) if the group mask is nonzero. Otherwise, it restores the user group affinity of the thread. |
| **KeSetSystemAffinityThread** | Changed | Sets the system group affinity of the current thread to group 0 and sets the specified group-relative affinity (in KAFFINITY). |
| **KeSetSystemAffinityThreadEx** | Changed | Sets the system group affinity of the current thread to group 0, sets the specified group-relative affinity (in KAFFINITY), and returns the previous group-relative system affinity of the thread (in KAFFINITY). The function does not return the previous group number. |
| **KeSetSystemGroupAffinityThread** | New | Sets the system group affinity of the current thread to the specified group affinity (in GROUP\_AFFINITY) and returns the previous system group affinity (in GROUP\_AFFINITY). |
| **KeSetTargetProcessorDpc** | Changed | Sets the DPC target to the corresponding logical processor in group 0, if the specified target processor number is less than the number of active logical processors in group 0. Otherwise, the DPC target does not change. |
| **KeSetTargetProcessorDpcEx** | New | Sets the DPC target to the logical processor number that is specified by a PROCESSOR\_NUMBER structure. |
| **NtQueryInformationProcess** | Changed | The new **ProcessGroupInformation** information class returns the list of groups (in an array of USHORTs) that are assigned to the process.  Note that the **ProcessAffinityMask** information class is not supported in Windows 7 or in earlier versions. |
| **NtQueryInformationThread** | Changed | The new **ThreadGroupInformation** information class returns the group affinity (in GROUP\_AFFINITY) of the thread.  Note that the **ThreadAffinityMask** information class is not supported in Windows 7 or in earlier versions. |
| **NtQuerySystemInformation** | Changed | The following information classes return the per-processor data for all logical processors in the group to which the calling thread is assigned:   * **SystemProcessorPerformanceInformation** * **SystemProcessorPerformanceDistribution** * **SystemProcessorPowerInformation** * **SystemProcessorIdleInformation** * **SystemProcessorIdleCycleTimeInformation** * **SystemProcessorCycleTimeInformation** * **SystemInterruptInformation**   The following information class returns the per-processor data for all logical processors in group 0.   * **SystemLogicalProcessorInformation** |
| **NtQuerySystemInformationEx** | New | Returns the following types of information for each logical processor in the specified group:   * **SystemProcessorPerformanceDistribution** * **SystemProcessorPerformanceInformation** * **SystemProcessorPowerInformation** * **SystemProcessorIdleInformation** * **SystemProcessorIdleCycleTimeInformation** * **SystemProcessorCycleTimeInformation** * **SystemInterruptInformation** * **SystemLogicalProcessorInformation** |
| **NtSetInformationProcess** | Changed | Changes as follows:   * The **ProcessAffinityMask** information class sets the process affinity (in KAFFINITY) if the process is assigned to a single group. If the process is assigned to more than one group, this function fails. * The new **ProcessGroupInformation** information class assigns a list of groups (in an array of USHORTs) to the process. |
| **NtSetInformationThread** | Changed | Changes as follows:   * The **ThreadAffinityMask** information class sets the thread affinity (in KAFFINITY) of the thread in the group to which it is currently assigned. * The new **ThreadGroupInformation** information class assigns the group affinity (in GROUP\_AFFINITY) to the thread. |

# Resources

Win32 and COM Development on MSDN

Process and Thread Reference  
<http://msdn2.microsoft.com/en-us/library/ms684852(VS.85).aspx>

InitializeProcThreadAttributeList Function  
[http://msdn2.microsoft.com/en-us/library/ms683481(VS.85).aspx](http://msdn2.microsoft.com/en-us/library/ms683481(VS.85).aspx%20)

System Information  
[http://msdn2.microsoft.com/en-us/library/ms724951(VS.85).aspx](http://msdn2.microsoft.com/en-us/library/ms724951(VS.85).aspx%20)

Windows Driver Kit

Interrupt Affinity and Priority  
<http://msdn.microsoft.com/en-us/library/aa489426.aspx>

IoConnectInterruptEx  
<http://msdn.microsoft.com/en-us/library/aa490464.aspx>

KeSetTargetProcessorDpc  
<http://msdn.microsoft.com/en-us/library/ms801939.aspx>