

How PAE X86 Works

Article10/08/2009

Applies To: Windows Server 2003, Windows Server 2003 R2, Windows Server 2003 with SP1, Windows Server 2003 with SP2

How PAE-X86 Works

In this section

- PAE X86 Architecture
- PAE X86 Processes and Interactions

Physical Address Extension (PAE) X86 is one of the scale up technologies in the Windows Server 2003 family. The Microsoft operating environment delivers scalability through two methods: scale up and scale out. Scaling up refers to the ability to incrementally add system hardware resources (processors and memory) to increase overall system performance. Scaling out refers to distributing the computing workload among multiple servers with the ability to add or subtract servers to increase or decrease capacity.

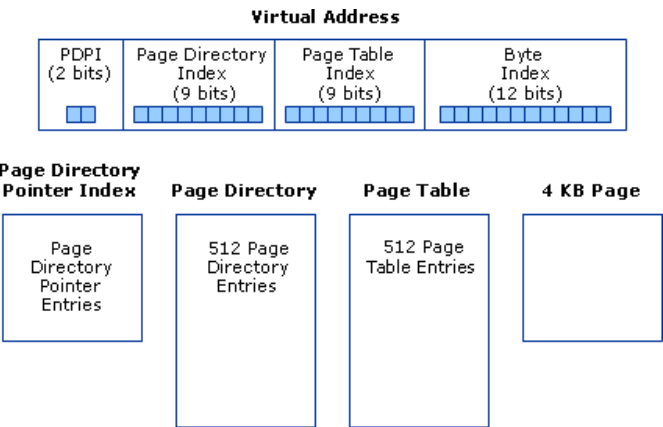
Enterprises are required to scale flexibly to respond to spikes in business and to support the rapid rollout of new products and services. It is therefore increasingly important that an organization’s computing infrastructure provides the ability to increase or decrease computing capacity almost instantly, while continuing to deliver on value. PAE X86 helps provide this scalability.

Note

- PAE X86 is not required on the 64-bit versions of the Windows Server 2003 family.


PAE X86 Architecture

PAE X86 alters the way the memory manager translates memory addresses. The memory manager moves from a two-level linear address translation to a three-level address translation. The extra layer of translation is what provides access to physical memory beyond 4 gigabytes (GB). The components used in the PAE X86 method of translation are shown in the following diagram.



The following table describes the components related to the PAE X86 architecture.

Components of the PAE X86 Architecture

 Expand table

Component	Description
Virtual Address	A virtual address that is translated into a specific physical address. This is the address that an application or the kernel uses when accessing data in memory.
Page Directory Pointer Index (PDPI)	A table listing pointers to specific page directories.
Page Directory	A table with 512 entries, each pointing to a specific page table.
Page Table	A table with 512 entries, each pointing to a specific 4 kilobyte (KB) portion of memory.
4 KB Page	A 4 KB portion of physical memory.

PAE X86 Processes and Interactions

PAE X68 is not automatically enabled. It must first be enabled by adding the `/PAE` switch to the `Boot.ini` file. By using this switch, the operating system loads a different kernel from `\i386\Driver.cab` file. The kernel file is `Ntkrnlpa.exe` for uniprocessor systems and `Ntkrpamp.exe` for multiprocessor systems. The kernel is loaded based on only the switch and independently from the existing physical memory.

Physical memory is treated as general purpose memory, so no new APIs are needed for the kernel to access the memory above 4 GB. Direct I/O can be done to addresses above 4 GB. This requires Dual Address Cycle (DAC) capable or 64-bit Peripheral Component Interconnect (PCI) devices. Devices and drivers offering this capability are called Large Memory Enabled (LME).

Because the 32-bit editions of Windows Server 2003, Enterprise Edition, and Windows Server 2003, Datacenter Edition, do not have a kernel PAE or LME API interface, the PAE X86 kernel ensures that many items are identical to the standard kernel, including:

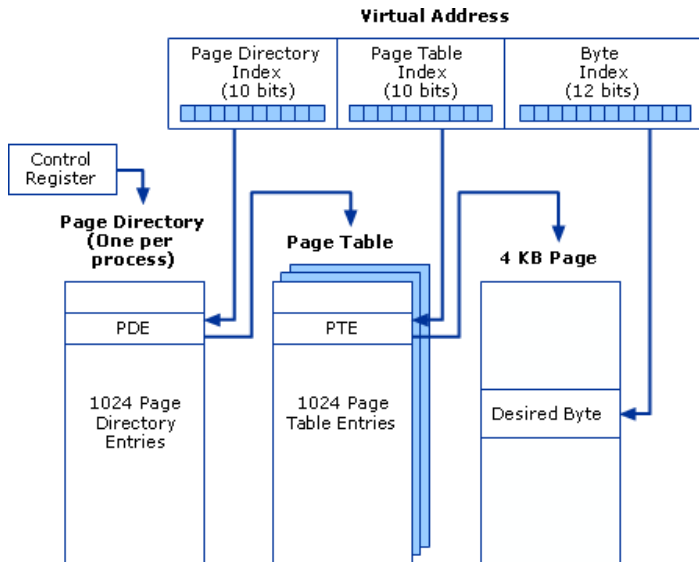
- Kernel memory space organization is unchanged.
- Registry flags work the same.
- Non-paged pool size remains the same.
- The 4-gigabyte tuning (4GT) feature is supported for up to 16 GB of RAM.
- The application header `IMAGE_FILE_LARGE_ADDRESS_AWARE`, which allows applications to make use of 4GT, will continue to work.
- “Well-known” kernel addresses remain in the same locations.

Applications do not need to be aware of PAE X86. Once enabled, the kernel will automatically make use of the additional memory for file caching, potentially improving the performance of applications running on the server.

Virtual Address Translation without PAE X86

When an application or the kernel requests access to physical memory, the memory manager must first translate the request, from a virtual address to a physical address. The normal implementation of virtual address translation uses a

two-level page table structure. When PAE X86 is not enabled, the 32-bit virtual address is interpreted as three separate components: a 10-bit Page Directory Index, a 10-bit Page Table Index, and a 12-bit Byte Index. The following figure shows the translation of the linear virtual memory address to a physical memory location when PAE X86 is not enabled.

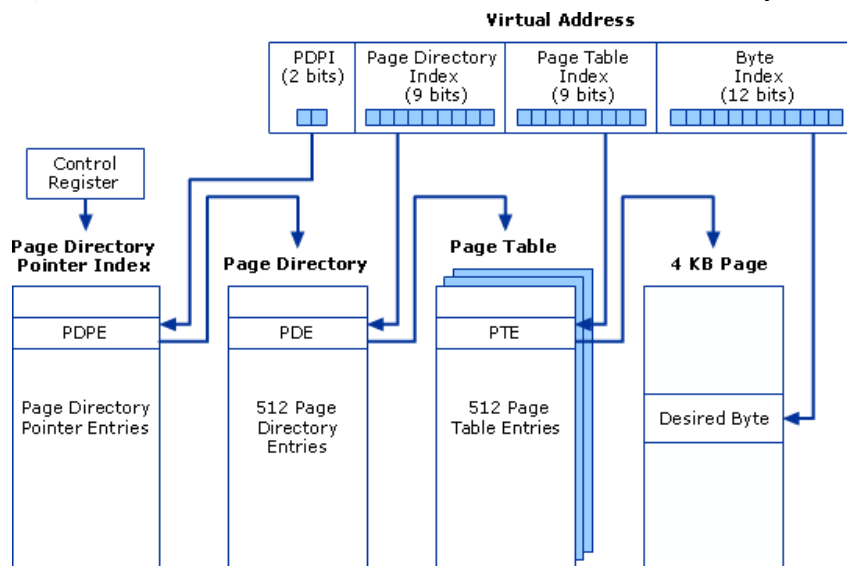


The following actions are involved in translating a virtual address when PAE X86 is not enabled:

- The memory manager locates the Page Directory for the current process. The memory manager is informed of the location of the Page Directory for the process by a special control register.
- The 10-bit Page Directory Index in the virtual address is used to locate a Page Directory Entry (PDE) that defines the location of the Page Table needed to translate the virtual memory address.
- The 10-bit Page Table Index in the virtual address is used to locate the correct Page Table Entry (PTE).
- The PTE is used to locate the correct 4 KB page of memory.
- After accessing this page, the 12-bit Byte Index in the virtual address is used to locate the physical address of the desired data.

Virtual Address Translation with PAE X86

With PAE X86 enabled, the operating system moves from a two-level linear address translation to a three-level address translation. The memory manager still implements Page Directories and Page Tables, but a third level of translation, the Page Directory Pointer Table, is also added. The extra layer of translation is what provides access to physical memory beyond 4 GB. The following figure shows the translation of the virtual address to a physical memory location when PAE X86 is enabled.



Instead of splitting the virtual memory address into three separate components for indexing into memory tables, it is split into four separate components: a 2-bit Page Directory Pointer Index (PDPI), two 9-bit components (a Page Directory Index and a Page Table Index), and a 12-bit Byte Index.

The control register now points the memory manager to the Page Directory Pointer Index (PDPI) that is 2-bits wide. The first two bits of the virtual address are used as an index into the table, with the resulting value pointing to a Page Directory. The first 9-bit field in the virtual address is then used to index the Page Directory. The indexed value then points to a Page Table. The second 9-bit field is an index into the Page Table. The resulting Page Table Entry (PTE) points to the actual 4 KB page in memory where the desired byte is located. The remaining 12 bits of data in the virtual address point to the specific byte of data on that page.

Page Table Entries

As described earlier in this section, the Page Table is used by the memory manager as part of the translation process between virtual memory addresses and physical memory addresses. When using the large amounts of memory made available through PAE X86, the number of PTEs needed to track that physical memory can increase dramatically. This can lead to problems when the amount of physical memory allocated to the kernel for storing PTEs is reduced, such as when using 4GT. With 4GT, the smaller amount of virtual memory allocated to the kernel means that the number of PTEs that can be recorded by the memory manager is much smaller. In a standard configuration (without 4GT), there are typically 80,000 to 140,000 PTEs available. This is reduced to only 40,000 PTEs when using 4GT. The next section describes the implications of this change.

PAE X86 and 4GT

4GT is another technology that makes more physical memory available to user applications. PAE X86 and 4GT can be combined to provide applications large amounts of virtual memory and provide a significant performance boost. However, using 4GT reduces the amount of PTEs available to the kernel, while using PAE X86 dramatically increases the amount of memory that must be indexed and translated by the memory manager. This combination will exhaust system kernel space much earlier than normal. Because of this, the memory manager imposes a virtual memory limit of 16 GB on a system with both 4GT and PAE X86 enabled. Even if a system has 32 GB or more of physical memory, if both options are enabled, only 16 GB of memory will be recognized. Thus, if PAE X86 is enabled on a system with more than 16 GB of physical memory, 4GT should not be used.

Note

- Even though the memory manager imposes a hard limit of 16 GB when both 4GT and PAE X86 are enabled, it is possible to encounter problems with lesser amount of memory, such as 8 GB or 12 GB. Therefore, the kernel should be given as much memory as possible.

Application Windowing Extensions (AWE)

Virtually all applications can benefit from the increased performance resulting from the kernel's use of PAE X86. However, in order to directly access the additional physical memory enabled by PAE X86, applications must use the Application Windowing Extensions (AWE) API set specifically designed for this purpose.

AWE is an API set that complements PAE X86. Unlike PAE X68, AWE is not an option set at startup and applications must directly invoke the AWE APIs in order to use them. Applications must be developed using the AWE API set in order to benefit from it.

Because an application's virtual address space can only be extended to 3 GB (using 4GT) on a 32-bit operating environment, applications (and database applications in particular, since they use large data sets and do a lot of swapping of data from disk to memory) need a method to map large portions of data into memory and keep that data in real memory at all times in order to increase performance.

AWE provides applications with this ability. AWE allows an application to reserve portions of real memory that cannot be paged out to the paging file or otherwise manipulated except by the reserving application. AWE will keep this data in real memory at all times. Because the memory manager does not manage this memory, it will never be swapped out to the paging file. The application is completely responsible for handling the memory.

AWE places the following restrictions on how this additional memory may be used, primarily because these restrictions allow extremely fast mapping, remapping, and freeing. Fast memory management is important for these potentially enormous address spaces and for attaining the highest possible performance.

- Virtual address ranges allocated for AWE are not sharable with other processes (and therefore not inheritable). Similarly, two different AWE virtual addresses within the same process are not allowed to map the same physical page. These restrictions provide fast remapping and cleanup when memory is freed.
- Since this memory is never paged, the physical pages that can be allocated for an AWE region are limited by the number of physical pages present on the computer. The memory is reserved until the application explicitly frees it or exits. The physical pages allocated for a given process can be mapped into any AWE virtual region within the same process. Applications that use AWE must be careful not to take so much physical memory that they cause other applications to page excessively or prevent creation of new processes or threads due to lack of resources.
- AWE virtual addresses are always read/write and cannot be protected. Read-only memory, no access memory, guard pages, or other similar calls cannot be specified.
- AWE address ranges cannot be used to buffer data for graphics or video calls.
- An AWE memory range cannot be split, nor can pieces of it be deleted. Instead, the entire virtual address range must be deleted as a unit when deletion is required.
- Applications can map multiple regions simultaneously, provided they do not overlap.
- Applications that use AWE are not supported in emulation mode. That is, an x86-based application that uses AWE functions must be recompiled to run on another processor, whereas most applications can run without recompiling under an emulator on other operating environments.

Paging File

Page files store data that cannot be kept in RAM. The default page file size is equal to 1.5 times the amount of physical memory on the system. On a 4 GB computer, this can result in a page file larger than 4 GB. Under such circumstances, the effectiveness of this large of page file could be minimal, based on the usage of the system, such as when hosting database management applications. However, there are also times when even larger page files are needed, such as when multiple applications are consolidated on to one system. Therefore, on servers with 4 GB or more of memory, it is often preferable to change the default size of the page file based on the usage of the server. Regardless, on x86-based computers with 4 GB or more of physical memory there should be a page file sufficiently large to capture a kernel memory dump.

A page file size of at least 2,050 megabytes (MB) is the minimum size necessary for computers with 4 GB or more of physical memory to capture a complete kernel memory dump. The page file can be much larger (up to three times the physical RAM is typical) if the system is being used for application consolidation.