Licensed Memory in 32-Bit Windows Vista

Though machines with 4GB are not yet the typical purchase for home or business use, they are readily available from major manufacturers and it won't be long before they are the typical purchase. But there are problems. You don't have to stand for long in a computer shop to hear a sales assistant talk of 4GB as some sort of limit for 32-bit operating systems, and it won't be long before this sales patter develops into outright promotion of 64-bit Windows as the only way to get past this limit. Some sense of this can be seen already in manufacturers' advertising materials, as in the following fine print from Dell:

The total amount of available memory will be less than 4GB. The amount less depends on the actual system configuration. To fully utilise 4GB or more of memory requires a 64-bit enabled processor and 64-bit operating system, available on selected systems only.

Let me stress now that I do not complain about Dell's statement. Its first two sentences are correct for all 32-bit editions of Windows Vista exactly as configured by Microsoft and installed by Dell. In the last sentence, I might quibble that the talk of a 64-bit processor is superfluous since the machine on offer does have such a processor, but otherwise the sentence is correct because of the word fully. Yet although Dell's statement is true, it is not the whole truth: there is something that Microsoft does not tell you, and perhaps does not tell Dell.

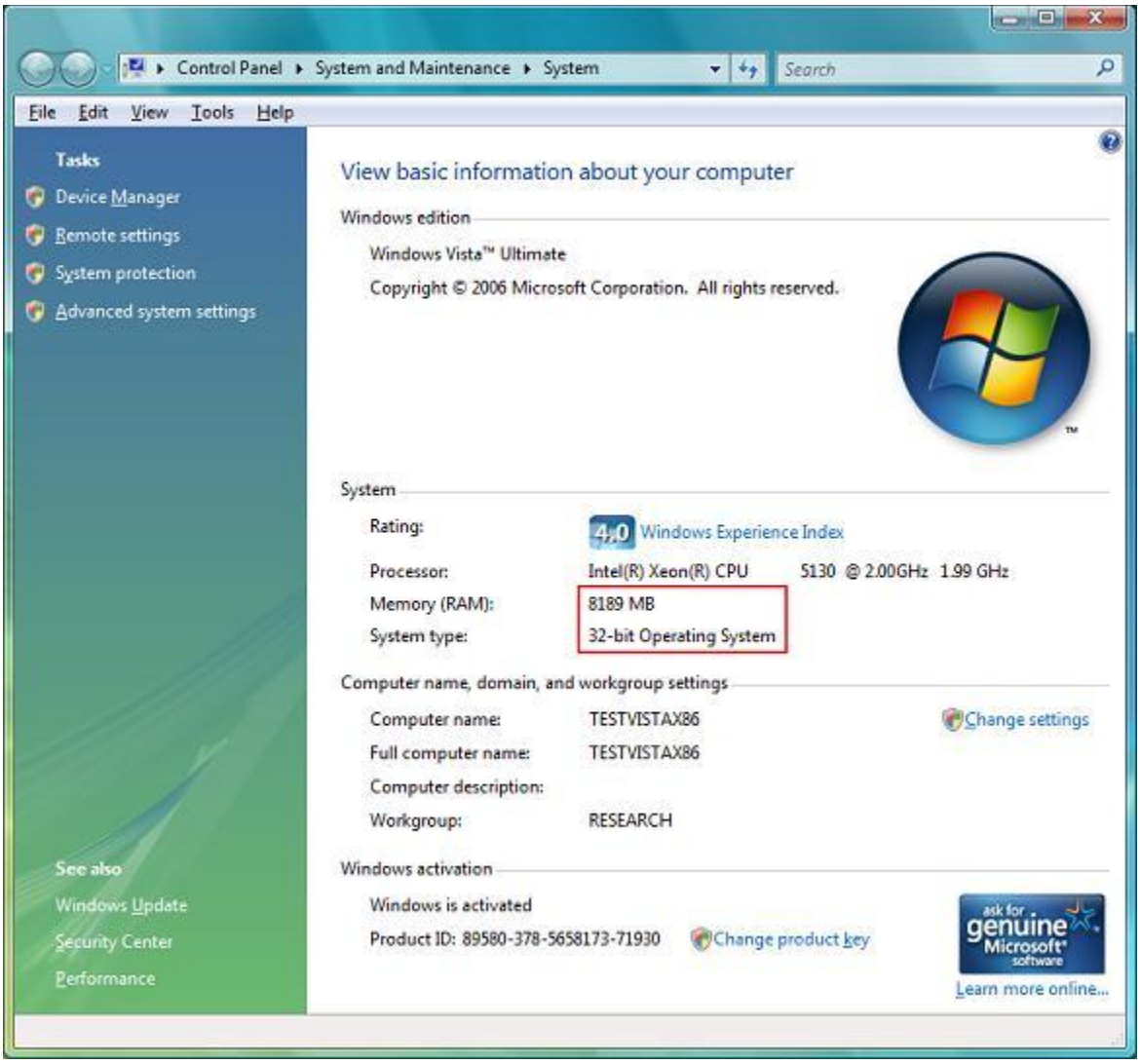
That 32-bit editions of Windows Vista are limited to 4GB is not because of any technical constraint on 32-bit operating systems. The 32-bit editions of

Windows Vista all contain code for using physical memory above 4GB. Microsoft just doesn't license you to use that code.

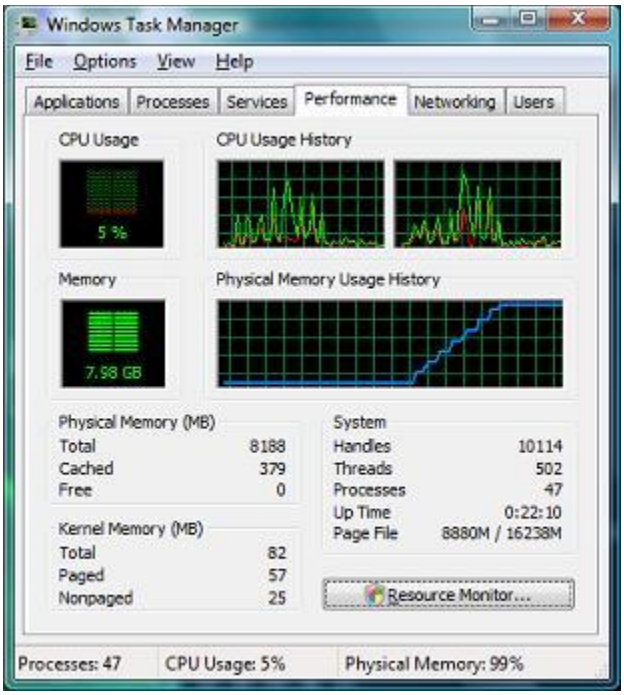
Well, to say it that way is perhaps to put words in Microsoft's mouth. I say the restriction to 4GB is a licensing issue because that's how Microsoft's programmers evidently have thought of it. The 4GB limit is retrieved from the registry by calling a function named ZwQueryLicenseValue, which is itself called from an internal procedure which Microsoft's published symbol files name as MxMemoryLicense. If you remove this check for the licensed memory limit then a restriction to 4GB is demonstrably not enforced by other means. Yet I must admit that I have not found where Microsoft says directly that 32-bit Windows Vista is limited to 4GB only by licensing. The supposed License Agreement doesn't even mention the word memory. What, really, is going on?

Demonstration

Put aside for now the fine print of what it means to "fully utilise" and ask what's even possible. Especially if you're one of the many who believe that 32-bit operating systems can't by definition use more than 4GB of RAM, what do you expect to see for the System Properties in the original 32-bit Windows Vista on a machine with 8GB of RAM? Click on the snapshot if you want it full-size and hi-fi:



No, this image is not a mock-up, though the red rectangle is my addition to highlight that this 32-bit operating system which ordinarily finds only 3069MB of RAM on this machine seems happy to have 8189MB. Windows will use all this memory, too, not that I have any ordinary need for it to do so. The next picture is as much a record of my unimaginativeness as of 32-bit Windows Vista actually using (very nearly) all the installed 8GB. An entirely ordinary test program writes 1GB of data from a single memory block to a file and reads the file back into that same memory block. That takes a while, even on a fast machine. By the time that eight instances are running concurrently, all the physical memory is in use:



Of course, there are contrivances and caveats. To get these pictures without contrivance, you would need a license upgrade from Microsoft, which Microsoft shows no sign of offering. The license data that would have to be upgraded is protected in various ways from being tampered with, and I certainly do not mean for anyone to try changing it, even for testing. Instead, to simulate having new license data from Microsoft, I have modified the kernel just enough so that it ignores the two license values that set memory limits, and I have started Windows in Test Mode so that it tolerates a kernel that no longer has Microsoft's digital signature. Neither of these steps is meant for general use. I am not solving for you the problem of how to have 32-bit Windows Vista use all your 4GB or more of physical memory without Microsoft's permission. Neither am I saying that 32-bit Windows Vista is a better (or worse) option than 64-bit for using all your 4GB of memory. I am just demonstrating that despite what Microsoft and many others say to the contrary, 32-bit Windows Vista actually can use all the RAM that you can feasibly install. The code for doing it is already in the product that Microsoft sells you. There is no need to bring in any code from a different edition of Windows or to make 32-bit Windows Vista believe it is anything other than 32-bit Windows Vista. All that needs to be changed are two pieces of data whose sole purpose is to specify how much memory Microsoft permits you to use. No code needs to be changed even by one byte, but to prove this point I have to patch the code because Microsoft does not permit changing the license data. If you want that this should work for you without contrivance, then pester Microsoft for an upgrade of the license

data or at least for credible, detailed reasoning of its policy for licensing your use of your computer's memory in 32-bit Windows Vista.

As for caveats, you should know that although I study software, I do not have the testing resources of a software or hardware manufacturer. I don't know of anything that misbehaves now that Windows finds all this memory that it ordinarily overlooks, but I can't swear that everything works correctly in every detail. With one exception, my test installation is just the original 32-bit Windows Vista Ultimate from MSDN discs, with all Windows features installed, but with no real-world additions of applications or drivers that aren't distributed with Windows or of devices that didn't come with the machine. Applications ought not matter, as argued below, but drivers may. This is where my one exception comes from: some of my tests failed until I updated the display driver (from NVIDIA). That's as good a reminder as any that when I say 32-bit Windows Vista has working code for using memory above 4GB, I talk of what Microsoft has written for Windows, not of what you add to your Windows installation from who knows where.

Still, even with contrivances and caveats, how can it be that I—or you after following my directions if you want to test what I say—can get anywhere near to producing such pictures? After all, for the page The system memory that is reported in the System Information dialog box in Windows Vista is less than you expect if 4GB of RAM is installed, Microsoft states plainly that "for Windows Vista to use all 4GB of memory … an x64 (64-bit) version of Windows Vista must be used." The pictures above show just as plainly that this statement by Microsoft cannot be entirely truthful. Even if you're perfectly happy to upgrade to 64-bit Windows Vista—and for all you're to know from reading this article, I'm among you on that point—I ask that you focus on whether Microsoft is open and truthful, and on whether the rest of the industry has been suitably vigilant and responsible. Even if you don't care about that for this issue, you may for another. The more that technology companies get away with half-truths and the exploitation of ignorance even on one issue, the more they can drift into it as their standard practice.

What is the truth, then? When someone says some such thing as that 32-bit Windows Vista is technically, physically, logicallly, architecturally, fundamentally or otherwise incapable of using all your 4GB or more of RAM, what can they mean?

There is already on the Internet and elsewhere an awful lot of rubbish to read about this question. Hardly any of it would be worth citing even if I didn't want to spare the authors the embarrassment. A surprising number of people who claim some sort of attention as expert commentators would have

you believe that using more than 4GB of memory is mathematically impossible for any 32-bit operating system because 2 to the power of 32 is 4G and a 32-bit register can't form an address above 4GB. If nothing else, these experts don't know enough history: 2 to the 16 is only 64K and yet the wealth of Microsoft is founded on a 16-bit operating system that from its very first version was designed to use 640KB of RAM plus other memory in a physical address space of 1MB. Some remember this history and add seemingly plausible qualifications that exceeding 4GB is possible only at the price of nasty hacks that require everyone—well, all programmers—to jump through hoops. Fortunately, Intel's processors are a lot more advanced than the 8086 from all those years ago.

Physical Address Extension

Old hands may already have groaned at the preceding heading. The means for a 32-bit operating system to use physical addresses above 4GB was built into Intel's 32-bit processors well over a decade ago[1] and has been supported by Microsoft since Windows 2000. If you haven't heard of it, or haven't thought that it applies to Windows Vista, then one reason may be that Microsoft has mostly advertised it only as a feature of the server editions such as Windows 2000 Server and Windows Server 2003, and only then for the more expensive levels with names like Enterprise and Datacenter. However, even Windows 2000 Professional can be configured, without contrivance, to access memory above 4GB by using Physical Address Extension (PAE). This is old technology. It's also widely and deeply misunderstood technology, arguably more than any other in the history of personal computing.

The essence of PAE is that the 32-bit registers used by 32-bit instructions in a 32-bit operating system do not in practice address physical memory. This is because of very old technology called paging. From at least as long ago as Windows 3.0 Enhanced Mode established Windows as the operating system to replace DOS, no software running on a 32-bit operating system (except very early during system initialisation) gets to address memory directly. The 32-bit register with which a program or driver or the operating system itself addresses memory holds what is called a linear address.[2] The processor translates linear addresses to physical addresses by looking through page tables, which are configured by the operating system. The layout of linear address space need have nothing to do with the layout of physical address space. This is how a DLL, for instance, can be loaded at an address not far below 2GB even on a machine that has only a few megabytes of RAM. Pages are typically small, just 4KB each. Two neighbouring pages in linear address space can come from opposite ends of physical memory. It's all up to the

operating system's memory manager and almost all of us take this paging mechanism completely for granted.

For the 80386 in 1985, each page table entry (PTE) was 32 bits and allowed for a 32-bit physical address. However, there is nothing fundamental to that. What's fundamental is only that every linear address must either map to a physical address or be not-present. There is no reason at all that the linear and physical address spaces must be the same size. With a suitably different translation algorithm, the physical address space can be as big as Intel wants to allow. This theoretical point, which I expect was appreciated at Intel from the outset, got its real-world implementation in the P6 family of processors, beginning with the Pentium Pro in 1995. Since then, with only very few exceptions, Intel's processors that are suitable for running 32-bit Windows all have enough address lines for accessing 64GB of memory and all support a translation algorithm for using all that memory in 32-bit code. PAE is this alternative translation algorithm.

The practical outcome for 32-bit operating systems in general is that although any one instruction can form addresses for only 4GB of linear address space, those 4GB can be drawn together page by page from all over any size of physical address space. For Windows in particular, the design is that the linear address space changes for each process. In 32-bit Windows, a process's user-mode code is allowed between 2GB and 3GB of linear address space (depending on the increaseuserva boot option), and the remainder of the 4GB is reserved for use by kernel-mode code. Both 32-bit and 64-bit Windows can use all of physical memory, including above 4GB, but a 32-bit Windows application has at most 3GB of linear address space through which to access any physical memory.

The difference between that and the "fully utiltise" in Dell's fine print seems very fine to me, especially while I don't have any real-world applications that need (or even use) as much as half a GB for each running instance. Until software that uses memory by the gigabyte becomes common for ordinary use outside of specialised contexts, this difference from full utility does not of itself justify a rush to 64-bit operating systems—and certainly not of disturbing a working, trusted installation of 32-bit Windows Vista. If you have a 32-bit program that wants more than its 2GB or 3GB, then upgrading to a 64-bit version of that program to run on a 64-bit operating system is your only path ahead. If you're buying a new computer and new applications, then getting 64-bit Windows and 64-bit applications is obviously the way of the future. Meanwhile, if your concern is only that the system and all your 32-bit applications may together use all your 4GB or more, then keeping your 32-bit operating system would at least be an option

for you if Microsoft would provide you with license data to let you "fully utilise" the PAE support that Microsoft has already coded into the product.

PAE Is An Ugly Hack?

Some commentators seem to have trouble grasping the naturalness of a physical address space that is larger than the linear address space. Perhaps they have been distracted by paging's historical role as technology for dealing with a shortage of physical memory. Perhaps they have in mind the history of MS-DOS, which was kept alive for many years with ever more ways that programmers might write new code to access ever more memory than the basic 640KB.

PAE is nothing like that. It is no more a concern to any software than is paging. After all, it is nothing but a variant algorithm for paging. Just as hardly any software is concerned that linear addresses are translated to physical addresses, even less software is affected by how linear addresses are translated to physical addresses. Application-level code and even most system-level code is entirely unconcerned and unaffected. Except for the operating system's memory manager and for the relatively few drivers that work with physical memory addresses, most notably for Direct Memory Access (DMA), no 32-bit software needs any recoding to benefit from a more-than-32-bit physical address space.

Even kernel-mode drivers don't need to know anything specific to PAE, much less be written specially to support it. All that's required is a general awareness that physical memory addresses may be wider than 32 bits and that accommodation of this comes naturally from following the documentation. Far from being an ugly hack, PAE requires pretty much nothing of anyone. Indeed, to write a driver that misbehaves only when memory is present above 4GB, you actually have to work at it, either by programming artificially or by convincing yourself that you don't need to do all that the documentation spells out.

When working with physical memory addresses, device drivers need to do 64-bit arithmetic. This should be natural since Microsoft's development kit for device driver programming has recommended it for well over a decade, including to define a 64-bit PHYSICAL_ADDRESS type that is used by all functions that receive or return physical memory addresses. With only a few highly contrived exceptions, any errors with a 32-bit driver's handling of 64-bit physical addresses, e.g., to discard the high 32 bits, are as much in error if left unfixed in the same device's 64-bit driver for 64-bit Windows.

For the particular matter of working with DMA, device drivers need to conform to the long-documented functional requirements for setting up and managing their DMA transfers. In particular, they need to be aware that the DMA functions may succeed only partially, and need to be called again to complete the request. The most significant, but not the only, reason for partial success is that the necessary double buffers could not all be set up. Double buffering is a technology for when a device cannot handle the full range of possible physical memory addresses. For instance, an old type of device (such as a floppy disk drive controller) may be limited to 24-bit physical addresses. To get data from the controller to physical memory above 16MB, the driver must use the DMA functions properly, so that the controller actually reads to a double buffer below 16MB and the DMA functions then copy the data to where it was wanted. A less old type of device (such as an IDE controller) may be limited to 32-bit physical addresses and will need double buffering in any operation that reads or writes to memory above 4GB. Of course, most devices can handle 32-bit physical addresses and increasingly many can handle 64-bit addresses. Either way, their drivers are supposed to use the DMA functionality as if double buffering may turn out to be needed. Some drivers for 32-bit Windows assume that since all physical addresses fit 32 bits, their 32-bit device needs no double buffering. They then take shortcuts with their use of the DMA functions. If these drivers are not fixed, then using physical memory above 4GB will expose the liberty that they have taken with the documented coding model. Note that if the device can handle 32-bit physical memory addresses but not 64-bit, then its driver needs to be fixed for 64-bit Windows, too.

None of this is to say that drivers do not exist whose faults are exposed when PAE enables use of memory above 4GB, or even that they never existed in any significant number, but it is to say that the main types of fault must be confronted in the development of a 64-bit driver for the same device, so that retention of these faults in the contemporaneous 32-bit driver is highly implausible. If you are worried that 32-bit Windows Vista with PAE may be unsafe because of faulty 32-bit drivers (or inadequate hardware, for that matter), then you would do well to wonder how 64-bit Windows can be any less unsafe on the same machine.

PAE and Performance

Some commentators say that PAE comes at some hideous cost to performance. Compared with the original algorithm that maps 32-bit linear addresses to 32-bit physical addresses, PAE is slower. It has one extra level to its page tables. Each PTE is twice as big. The operating system therefore

has more work to do when preparing and maintaining the page tables, and since the Translation Lookaside Buffer (TLB) has only half the capacity, memory references are more likely to miss the TLB and require additional bus cycles. The reduction in performance is surely measurable. If you have no need to access memory above 4GB and are concerned enough, then you would not enable PAE. Note however that Microsoft does not regard this performance cost as worth troubling over (as will be clear shortly, under the heading Data Execution Prevention).

Of course, for access to memory above 4GB, the appropriate comparison is not between using PAE and not, but between using PAE and using 64-bit Windows. For this comparison, not only are the PTEs the same size but the algorithms are very similar. To the processor, it's PAE that is slightly simpler and plausibly quicker, but the memory manager in a 64-bit operating system can benefit from using 64-bit registers when working with the PTEs. These are very fine trade-offs relative to the enormous overheads that embellish some of the wilder misunderstandings of PAE on the Internet.

For a rough-and-ready assessment of these trade-offs, consider Microsoft's own performance measurement, as given by the Windows Experience Index. Surely this is meant to have some objectivity, even if comparison of ratings for 32-bit and 64-bit Windows may not be strictly fair. On this article's test machine, the "Memory (RAM)" component of the Windows Experience Index is consistently 5.0 in 64-bit Windows Vista and is just as consistently 5.1 in 32-bit Windows Vista, whether PAE is enabled or not.

Choosing PAE

Whether the memory manager in the Windows kernel uses PAE is configurable through the pae boot option. Indeed, 32-bit Windows Vista is supplied with two kernels:

- an ordinary kernel which uses 32-bit PTEs without PAE, and has no code for working with physical addresses above 4GB;
- a PAE kernel which uses 64-bit PTEs with PAE, and does have code for working with physical addresses above 4GB.

The two kernels are respectively NTOSKRNL.EXE and NTKRNLPA.EXE, both in the Windows System directory. The loader (WINLOAD.EXE) knows how to set up the linear address space for mapping to physical addresses with or without PAE, but each kernel is specialised to one algorithm for the mapping. The pae option tells the loader which kernel to load.

Data Execution Prevention

If you have a modern machine of the sort that manufacturers are fitting with 4GB of RAM, then you very likely are running the PAE kernel already. This is not so that you can use physical memory above 4GB, else this article would not exist. It is instead to give you what Microsoft calls Data Execution Prevention (DEP). This protects you from programs that try to execute data, whether in error or from (suspected) malice. The connection with PAE is that DEP depends on the NX bit that AMD has defined (and Intel adopted) in 64-bit PTEs, such that DEP can only be enabled if PAE is also enabled. Because Microsoft wants you to benefit from DEP, the typical practice of Windows Vista is to select the PAE kernel if you haven't specified that you want it and even if you have specified that you don't want it. (If your machine supports DEP, then a necessary condition for disabling PAE is that you also disable DEP by setting nx to AlwaysOff.)

Physical Memory Map

That you have 4GB of RAM does not mean that all physical memory addresses from zero to 4GB actually do reach any RAM. In practice, much of that range of physical address space, most likely at the top, is given over to such things as the system BIOS and devices. You can get some sense of this by starting the Device Manager, opening the View menu and asking to see "Resources by type" or "Resources by connection" and then expanding Memory. What this gives you, however, is at best only an indication. It tells you that some addresses are used for devices. It doesn't tell you which addresses actually do have RAM (or ROM, for that matter).

The memory map that matters most for the question of what physical memory the kernel can use is the map that the loader discovers from the firmware. For machines whose firmware is a PC-compatible BIOS, the means of discovery is int 15h function E820h.[3] Unfortunately, the loader does not save this map exactly as learnt from the BIOS, which complicates your inspecting this memory map for yourself. However, Windows Vista introduces some undocumented functions with which a kernel-mode driver can get the map fresh from the BIOS. Such a driver for viewing the firmware memory map is presented separately, along with a small console application that reports the results. You will need administrative privilege to load the driver.

Of particular interest once you have the firmware's memory map for your computer are the ranges that are reported as RAM. This article's test

machine has its 8GB of RAM in four ranges spread through 9GB of address space:

Address	Size	Remarks
00000000`00000000	00000000`0009FC00	640KB of base RAM, less 1KB as an Extended BIOS Data Area
00000000`00100000	00000000`BFD0AC00	not quite 3GB at 1MB
00000001`00000000	00000001`00000000	4GB at 4GB
00000002`00000000	00000000`40000000	1GB at 8GB

The first 3GB of physical address space has RAM in two ranges because some is lost at the top of the first 1MB (for reasons of compatibility that go all the way back to the original IBM PC) and some more is lost at the end of the 3GB. The next 1GB is so much given over to device memory that instead of wasting RAM at 3GB, hardware remaps the RAM from there to the end of all other RAM, where it shows as the last of the ranges. The total amount of addressable RAM in the first 4GB is 3,143,338KB, i.e., 3069MB and 682KB. On this machine, with its present configuration of hardware, if the kernel is limited to a physical address space of 4GB, then 3069MB (and the spare change) is all the RAM that the kernel can possibly use. Get the kernel to recognise physical addresses above 4GB, and it picks up the other 5GB, for a total of 8189MB as shown in the picture.

If the 4th gigabyte were left at 3GB, Windows would have access only to as much of it as does not get overridden. In practice, RAM might show through in various gaps, so that the amount of RAM accessible below 4GB would be more than 3GB but nowhere near 4GB. If you have exactly 4GB of RAM installed, then getting the kernel to use physical addresses above 4GB will be no benefit to you unless some of your 4GB of RAM is remapped above the 4GB address. Whether this remapping is done at present on your particular machine can be checked by using the separately supplied driver. If it is not done, then whether it can be arranged is an issue of hardware configuration. Check your BIOS Setup, read your chipset manual, or consult your computer's manufacturer.

Of course, for a machine that has exactly 4GB of RAM and has 32-bit Windows Vista pre-installed, you would expect that the manufacturer, having been told by Microsoft that Windows will not see any RAM above 4GB, might not have configured any of the 4GB to be remapped out of sight and into uselessness. You should not be surprised to find that remapping is disabled. Worse, unless the manufacturer anticipates installing other Windows versions on the machine, there is no incentive even to provide for

remapping above 4GB as something that you can configure if you want. Indeed, it may even be that your chipset can't handle physical memory addresses that are wider than 32 bits. If so, then memory above 4GB isn't safe to use whatever your operating system (and you would hope your BIOS does not even think to report that any such memory is present). If your chipset does not support remapping, then RAM that is overridden for device memory below 4GB will never be seen as usable RAM by 32-bit Windows even with PAE enabled and is just as much lost to you if you install 64-bit Windows.

License Values

How does it happen, then, that 32-bit Windows Vista is supplied with a PAE kernel which is capable of using memory above 4GB but doesn't actually use any of that memory? Broadly speaking, there are two mechanisms by which memory above 4GB does not get used. One is that the kernel never learns of any such memory. The other is that the kernel knows the memory is there but deliberately ignores it.

The first of these mechanisms comes about from a boot option, named truncatememory, which tells the loader to discard all pages of physical memory that are not wholly beneath some specified address. Thereafter, the discarded memory may just as well never have been discovered from the firmware. When the kernel receives the memory map from the loader, the discarded memory is already (long) gone.

Of course, most people want all their computer's memory to get used, and so the truncatememory option is ordinarily not set (even by people who know it exists). By the time the kernel receives the memory map from the loader, the map has been much refined in order to account for how memory is already in use, but it is otherwise intact.[4] Very early during the kernel's initialisation, however, the kernel sets about its own filtering of this memory map. Limits are applied both to the total amount of usable memory and to the maximum physical address. Memory in excess of these limits is discarded, such that it may as well never have been passed to the kernel from the loader.[5]

Total Memory

The total amount of memory allowed is taken solely from the license value Kernel-WindowsMaxMemAllowedx86, as read through the undocumented function ZwQueryLicenseValue. The data for this value is a number of MB, so

that 0x1000, which is installed for all 32-bit editions of Windows Vista, means 4GB.

Maximum Address

The maximum physical address is calculated as the least of three values: a license limit; a run-time limit; and a hard-coded limit.

For the ordinary kernel, the license value for the maximum physical address is the same as for the total amount of memory, but the PAE kernel has a separate license value, named Kernel-MaxPhysicalPage. Again, the data for this value is a number of MB. Again, all 32-bit editions of Windows Vista are installed with this value set to 0x1000, representing 4GB.

The run-time limit arises from needing to be sure that an array of MMPFN structures can be set up to represent all the pages of physical memory, one structure per 4KB page, from zero up to the maximum physical address. The kernel's capacity for such an array depends on how much of the linear address space is already in use. This gives the run-time limit on the maximum physical address. It is the size of the largest block of free linear address space, divided by the size of an MMPFN structure (0x18 or 0x1C bytes for the ordinary and PAE kernels, respectively), and then multiplied by the page size.

The preceding calculation also produces an architectural limit on the use of physical memory by 32-bit Windows with a PAE kernel. The largest block of linear address space that is available even this early cannot be as large as 1GB and could be much smaller. Even if it is very nearly 1GB, that's only enough to fit an MMPFN array for a maximum physical address of 128GB (given that the maximum should be a power of two). This limit is hard-coded in the PAE kernel, as is 4GB for the non-PAE kernel.

For the question of whether the kernel in 32-bit Windows Vista will use all the physical memory it learns about from the loader, the hard-coded limit of 4GB is dominant as the maximum address for the ordinary kernel, which truly cannot form addresses for physical memory above 4GB, but the license limit is dominant for the PAE kernel. If you have physical memory above 4GB and wonder how it can be that the PAE kernel does not use that memory, the answer is licensing. The 32-bit code for using memory beyond 4GB is present in Windows Vista as Microsoft supplies it, but Microsoft prepares license values in the registry so that this code never gets to work with any physical addresses above 4GB.

Transparency

Microsoft is not exactly forward in describing this mechanism by which 32-bit Windows Vista is restricted to 4GB. Especially notable is that the page Memory Limits for Windows Releases doesn't mention the word license. Some explanation may be that Microsoft takes licensing so much for granted that it is simply left as understood that the stated limits on physical memory are licensing limits.

For the page The system memory that is reported in the System Information dialog box in Windows Vista is less than you expect if 4GB of RAM is installed, Microsoft again does not admit that all the necessary code for using memory above 4GB is present in the product, but does hint at it when saying that "the 32-bit versions of Windows Vista limit the total available memory" to protect users against incompatible drivers. Though no drivers need to know the mechanics of PAE and only a small proportion work at all with physical addresses, they may be indirectly incompatible with PAE because they have been coded to an assumption that the physical address space is the same size as the linear address space.

Accept for now that such incompatibilities with PAE have significant effects for significantly many users, and this page of Microsoft's is still disingenuous. Of course, with DEP enabled as the typical Windows configuration (starting with Windows XP SP2), Microsoft will have needed to limit Windows to the first 4GB of physical address space so that the benefits of DEP are not blown away by problems from third-party drivers that misbehave when memory is present above 4GB. But Microsoft's published reasoning would be satisfied by arranging that enabling DEP not only enables PAE but also adds by default a truncatememory option to stop the kernel from knowing of any memory above 4GB. It does not of itself explain why the restriction to 4GB has instead been implemented in a way that prohibits the use of memory above 4GB by everyone, even at their own risk or when using computers whose manufacturers test their machines and trust their drivers and are willing to bear the support costs if they're wrong.

Anyway, how significant could these incompatibilities be in real-world use of Windows Vista? The drivers that Microsoft talks about are not the sort of things that users install willy-nilly. They are much more the sort that come installed with a new computer, such that they have been tested (or ought to have been) by the manufacturer. They are also the sort of driver to which Microsoft will not give a digital signature unless the driver passes Microsoft's testing at the Windows Hardware Quality Labs (WHQL). Add that Microsoft's Device Driver Kit (DDK) for writing device drivers has defined the

PHYSICAL_ADDRESS type as 64 bits since at least Windows NT 4.0 and that double-buffering into physical memory below 4GB has been supported since Windows 2000, and you might be forgiven some incredulity that these incompatibilities can now exist in any number, let alone that concern for them explains the forcible crippling of new installations of 32-bit Windows Vista on new computers.

Moreover, as noted earlier, the main types of coding error that are exposed by using memory above 4GB on 32-bit Windows are as much a problem to 64-bit Windows. It would be not just incredible but implausible that these errors are retained in 32-bit drivers that have been ported to 64-bit. However significant may have been the problems of 32-bit drivers and hardware for the safe use of memory above 4GB by 32-bit Windows in earlier versions, the natural expectation must therefore be that they are rapidly being eliminated from real-world occurrence by the widespread adaptation of those drivers to support 64-bit Windows Vista. Add that any hardware that won't support 32-bit Windows with PAE won't support 64-bit Windows either, and it is a wonder that anyone, inside Microsoft or out, can keep a straight face while saying that an upgrade to 64-bit Windows Vista is obviously safe but enabling PAE on 32-bit Windows Vista (with up-to-date drivers) just as obviously isn't.

Even Microsoft looks to be uncertain of its ground when talking about these incompatibilities. In the article just cited, which appears to be Microsoft's main explanation of why 32-bit Windows Vista in particular is limited to using less than 4GB of memory, some paragraphs given as More Information are not even relevant to the question of using memory above 4GB. It simply does not matter that "DEP may cause compatibility issues" with any driver let alone with any "that performs code generation or that uses other techniques to generate executable code in real time." A problem from such a driver is resolved by disabling DEP, with no effect on using memory above 4GB. Why is that paragraph even present on Microsoft's page, if not to suggest a greater weight of argument to casual or uninformed readers?

Another paragraph that Microsoft presents as More Information is even worse. It talks of drivers that directly modify the page tables and "cause system instability" because they "expect 32-bit page table entries but receive 64-bit PTEs in PAE mode instead." Put aside that you, as a security-minded user, ought not want (and hopefully don't have) such drivers executing on your system even when you have 32-bit PTEs. Consider instead that these drivers will likely do the same mischief when PAE is enabled just for DEP. The PTEs are still 64-bit even if they never hold a physical address above 4GB. These errant drivers will still miscalculate the location of every PTE that they want to modify. If fear of this is an argument against using

memory above 4GB, then it is just as much an argument against enabling DEP (which Microsoft recommends should always be enabled).

As Microsoft's technical arguments for limiting 32-bit Windows Vista to 4GB of physical address space, these are strikingly poor in quality. They look more like the sort of arguments someone might pass off to rationalise a decision made on other grounds.

A Marketing Ruse

Perhaps the following, from Pushing the Limits of Windows: Physical Memory by Mark Russinovich at a Microsoft website, ends with a more frank description of Microsoft's thinking about this 4GB limit than can be found in anything written directly by Microsoft:

Because device vendors now have to submit both 32-bit and 64-bit drivers to Microsoft's Windows Hardware Quality Laboratories (WHQL) to obtain a driver signing certificate, the majority of device drivers today can probably handle physical addresses above the 4GB line. However, 32-bit Windows will continue to ignore memory above it because there is still some difficult to measure risk, and OEMs are (or at least should be) moving to 64-bit Windows where it's not an issue.

See that the difficult-to-measure risk is merely asserted despite an acknowledgement that it seems implausible for new computers. As with Microsoft's own literature, no comment is ventured about why this particular risk, among all the things that can go wrong with drivers, is so special that it must be handled as a licensing matter rather than letting users and manufacturers choose for themselves if they trust their hardware and drivers. It has the look of providing cover for moving consumers to 64-bit Windows faster than they might otherwise go. Just accept it without question and be glad for the new business as consumers install 64-bit Windows and start buying 64-bit applications. Who in the computer industry—whether a manufacturer of hardware or software, or even a commentator whom some might think is an independent analyst—is going to criticise Microsoft for a sleight of hand that brings forward a cycle of upgrades!

Windows With The Lot?

There is much that's unsatisfactory about Microsoft's hand-waving over imposing a 4GB limit to 32-bit Windows Vista. If nothing else, when consumers pay for a software product in an edition that the manufacturer

describes as Ultimate, they surely have a reasonable expectation that the software is licensed to do everything that its code is capable of. If you buy only the Home Basic edition instead of Home Premium, you expect to get less software and be licensed to use fewer features. Surely the point to an edition that is called Ultimate is that you get the whole package and are licensed to use it all. If you pay the extra for Ultimate but you're not licensed for everything that the software can do, then how is Ultimate ultimate?

Note that this is not a case of a manufacturer puffing up a product description. The complaint is not that Ultimate isn't truly ultimate because it isn't the last word in what naive consumers might imagine Ultimate could be if only Microsoft would write the code. It's that Ultimate deliberately is not given the ultimate license to do all that its code is already capable of.

Anti-Competitive Practices

By licensing Windows Vista only for memory below 4GB, Microsoft suppresses competition in the subsidiary market of computers that run Windows Vista. Given that Windows Vista is the current representative of a monopoly product, this suppression may count as an anti-competitive practice in terms of some jurisdictions' laws on monopolistic abuses. If 32-bit Windows could not use physical memory above 4GB because it has no code for using PAE, there would be no possible competition to suppress, and no possible grounds for complaint. Instead, since Windows actually does have code for using PAE, the manufacturers of peripheral devices, the programmers of the corresponding device drivers and the assemblers of computers could have competed among one another on whether their wares have the higher quality that Microsoft itself says is expected of 32-bit drivers for safe use of physical memory above 4GB.

An irony is that the very same driver incompatibilities that Microsoft talks of as a danger to users would surely have been eliminated—long ago—by the natural forces of competition among driver developers and device manufacturers had such competition not been stifled. Instead, Microsoft for many years misled those developers and manufacturers to believe that the 32-bit client editions of Windows were not enabled for PAE when in fact they were (see below, in the section titled Past).

Though Microsoft may not have benefited directly from suppressing competition in the driver market, consumers have evidently not benefited at all. Especially while 64-bit Windows Vista was not so readily available, very many consumers bought computers with 4GB of RAM but not the whole use of that RAM. True, they will have proceeded with their purchase despite

having seen the fine print that informed them of the loss, but they weren't fully informed, and they arguably have been misinformed about the reason that all their RAM can't be used.

You May Not Test What We Say

Especially unsatisfactory is that Microsoft says something about its product, and about other people's products, but uses the licensing mechanism to deny the means to test what's said. Whatever you think of software licensing, in general or as practised particularly by Microsoft, its use by any software manufacturer to frustrate independent testing of that manufacturer's claims about a consumer product is low.

When Microsoft and others make out that defective drivers are so widespread and dangerous that 32-bit Windows Vista cannot be allowed to use memory above 4GB even as a configurable option, how is anyone to know the truth of it? Nobody can test even one driver's use of memory above 4GB on 32-bit Windows Vista with the license data that Microsoft supplies for it. The closest that anyone can come is to test with 32-bit Windows Server 2008, which hardly any consumers ever get to see, and which of course was not available until some time after Windows Vista was first released. It is no credit to the computer industry that any manufacturer's arguments are accepted so widely without independent testing.

Testing the Use of Physical Memory Above 4GB

Even to begin to test whether a particular installation of 32-bit Windows Vista cannot safely use memory above 4GB because of driver incompatibilities, you must somehow side-step the two relevant license values.[6] For this purpose, it is enough just to modify the kernel so that where it presently reads the two relevant license values from the registry, it will instead proceed as if the license values had no effect.

Let me stress that although I have to modify the kernel, using memory above 4GB does not require a change to even one byte of code in 32-bit Windows Vista. That I modify any code here is merely to simulate the provision of different license data by Microsoft. Much as you can buy Windows Vista Home Basic and then upgrade to Home Premium without having to reinstall Windows, you might upgrade to a configuration in which the two license values for memory limits are raised. Because Microsoft protects those license values, this patch is as close to that upgrade as can

be arranged unless Microsoft makes the license upgrade available. If you patch the kernel and your tests then show to your satisfaction that your drivers are safe (though perhaps only after you disable defective drivers and install the latest updates from their manufacturers), then a license upgrade from Microsoft is what I intend you to seek.

You should understand now, i.e., before you get further involved, that patching the kernel presents some difficulties in excess of patching an application-level executable. Not only will you need a program with which to change a few bytes in the executable code, you will also need programming tools either to set the checksum in the executable's header or to sign code.

Patch Details

The only executable that is considered here is the PAE kernel, named NTKRNLPA.EXE, from 32-bit editions of Windows Vista (and Windows 7, which may as well be a Windows Vista service pack as far as concerns this article's subject). The known builds have a routine named MxMemoryLicense in which there are two sequences of nearly identical code, one for each relevant license value. Each sequence calls the undocumented function ZwQueryLicenseValue and then tests for failure or for whether the data that has been read for the value is zero. In the known builds, each sequence has the following instructions in common:

Opcode Bytes	Instruction
7C xx	jl default
8B 45 FC	mov eax,dword ptr [ebp-4]
85 C0	test eax,eax
74 yy	je default

At the time of executing, the eax register holds the status code from ZwQueryLicenseValue. The first instruction is the test for failure (indicated by a negative status code). The remaining instructions test whether the retrieved data is zero. In the known builds, the xx and yy placeholders are 0x11 and 0x0A respectively for the first sequence and 0x10 and 0x09 for the second, but even with the placeholders left unresolved, the sequence shown occurs just twice in the whole kernel. This may help you find the patch sites even if you have a different build (such as you may have picked up from Windows Update).

Both occurrences are to be patched the same way. The patch is designed to vary the ordinary execution as little as possible. The kernel is left to call ZwQueryLicenseValue as usual and to test for failure, but the last three of

the above instructions are changed so that the kernel proceeds as if the retrieved data is the value that represents 128GB (which is the least value that removes licensing from the kernel's computation of maximum physical address). Change the 7 bytes starting from 0x8B so that you now have the following instructions:

Opcode Bytes	Instruction
B8 00 00 02 00	mov eax,00020000h
90	nop
90	nop

The following table lists the known English-language builds of the PAE kernel and gives the file offsets for the two patch sites:

Version	Package	File Offsets
6.0.6000.16386	Windows Vista	0x003040B1, 0x003040F2
6.0.6001.18000	Windows Vista SP1	0x00309AA3, 0x00309AE4
6.0.6002.18005	Windows Vista SP2	0x0030C43A, 0x0030C47B
6.1.7600.16385	Windows 7	0x0035C243, 0x0035C283

For the remainder of these directions, I assume that your patched copy is named NTKR128G.EXE.

Checksum

For all executables that are loaded by WINLOAD, as is the kernel, the checksum in the executable's header must be set correctly. Since patching the kernel will almost certainly have invalidated this stored checksum, you need to reset it. Signing the code, as discussed under the next heading, will do this. If you don't have tools for signing code, then a suitable tool for setting the checksum is EDITBIN from Microsoft Visual Studio. Its /release switch exists solely to set the checksum. The command to run is

```
editbin /release ntkr128g.exe
```

Digital Signature

It is sometimes said that kernel-mode drivers are not checked for digital signatures in 32-bit Windows Vista, or more accurately that although hashes are computed, drivers are not rejected if the hash is not validated by a signature.[7] Although this is broadly true, there are a dozen executables that

the loader ordinarily insists be signed properly. The kernel is one of them, of course. It must be signed by a certificate that derives from one of a handful of root certficates whose public keys are hard-coded into the loader. Since patching the kernel will have invalidated Microsoft's signature, you have to resort to one of the exceptions that are covered by the word "ordinarily" (unless you want to patch the loader too).

One of these exceptions is a Test Mode which Microsoft provides so that drivers can be tested during development. Presumably, this Test Mode also lets Microsoft's own kernel programmers get their kernel tested while it is still being worked on and is not ready for a proper signature. In patching the kernel to test what will appear to you to be a new kernel-mode feature, you are in essentially the same position, though on a much smaller scale, as Microsoft's own kernel programmers when they have changed the kernel. In this sense, the Test Mode is the most appropriate way around the digital signature.

In Test Mode, the loader relaxes its integrity checking such that any root certificate is accepted.[8] Provided that you have suitable tools, you can create your own test-signing certificate and test-sign your modified copy of the kernel, such that it will load when you boot Windows with the testsigning option. There is a price however: Test Mode has the detraction of placing small warnings on the desktop.

For suitable tools, with documentation, look in either the Windows Software Development Kit (SDK) or the Windows Driver Kit (WDK). To make your own certificate, run some such command as

```
makecert -r -ss my -n "CN=My Own Testing Authority"
```

This creates a root certificate for an invented certification authority named My Own Testing Authority and installs it in the Personal certificate store, which is represented by "my" in the command. You can view the new certificate by starting the Certificate Manager (CERTMGR.MSC), which also lets you set a Friendly Name for the certificate if you want to keep it. To sign your modified kernel with this certificate, run the command

```
signtool sign -s my -n "My Own Testing Authority" ntkr128g.exe
```

Note that you do not need administrative privilege for these steps. Also, you can self-sign the kernel on one machine but test it on another. There is no need to transfer the certificate to the test machine. Indeed, there is no more need to keep the certificate. You can delete it from the Personal store either

through the user interface of the Certificate Manager or by running the command

```
certmgr -del -c -s my -n "My Own Testing Authority"
```

Booting the Self-Signed Patched Kernel

You now have a modified kernel with which to test your 32-bit Windows Vista for its use of physical memory above 4GB. Copy it to the Windows System directory of the machine that you will test. (For this, you typically will need administrative privilege for the target machine.) Then, provided that you successfully self-signed the kernel, you have to restart that machine with three boot options:

- pae, set to ForceEnable to be sure of enabling PAE;
- kernel, set to NTKR128G.EXE so that Windows will start with your patched kernel;
- testsigning, so that the loader will accept your self-signed kernel.

Persistent Test Mode

To set up a persistent test, use the BCDEDIT utility to write the options into the BCD store. It is prudent to work on a copy of the configuration that you want to test. Assuming that you are currently booted into this configuration, running the command

```
bcdedit /copy {current} /d "Windows Vista Using All My Memory"
```

will create a new entry for the boot menu and tell you a GUID which you then reproduce (e.g., by a copy and paste) in the commands:

```
bcdedit /set {guid} pae ForceEnable
bcdedit /set {guid} kernel ntkr128g.exe
bcdedit /set {guid} testsigning on
```

When you restart the computer, select "Windows Vista Using All My Memory" from the boot menu, and start testing. If you do turn out to have a defective driver and need to identify it and update it, then you can go through any of the usual processes of elimination even while starting in Test Mode. Note in particular that Test Mode is not Safe Mode. You can have the two together. While "Windows Vista Using All My Memory" is selected at the boot menu, just press F8 to open the Advanced Boot Options Menu and then select Safe Mode.

One-Time Test Mode

If you already have an operating system configuration that has PAE enabled, then a less intrusive way to start your tests is to enter the options at the Edit Boot Options Menu. This is admittedly a lot easier if your machine is already configured for booting multiple operating systems (or the one operating system in multiple configurations) so that you ordinarily see a boot menu during startup. At this boot menu, select the configuration that you want to test, then press F10 to open the Edit Boot Options Menu. It is too late to enable PAE here, but the other options can be added in the style of BOOT.INI switches:

```
/kernel=ntkr128g.exe  /testsigning
```

Press Enter, and the selected operating system will start with your modified kernel in Test Mode. This method has the advantage of not changing the selected operating system in any way that lasts, and the corresponding disadvantage that you have to type something every time you want to do the test.

Booting the Patched Kernel With a Bad Signature

The patched kernel can also be booted by disabling the loader's integrity checking altogether. You cannot set this up entirely with boot options in the BCD store, because the option to disable integrity checking is not permitted to persist. The closest you can get is to disable integrity checking at one or other of the boot menus.

Prepare a boot entry with just the pae and kernel options as above. When you restart the computer, select "Windows Vista Using All My Memory" but press F8 to open the Advanced Boot Options Menu and then select Disable Driver Signature Enforcement.

The alternative with the Edit Boot Options Menu is also available, with the same constraints about PAE being already enabled, but the switches to enter are now:

```
/kernel=ntkr128g.exe  /disable_integrity_checks
```

Curiously enough, booting with integrity checking disabled leaves no warnings on the desktop (such as you get from booting in Test Mode). I do not mean to recommend this method of testing. I describe it only for completeness and because it is the only method available if you do not have tools for code-signing, which are relatively new from Microsoft, but are able to set an executable's checksum.

Past

Although this article is motivated by the realisation that Microsoft introduced a formal scheme of license values, two of which stop 32-bit Windows Vista from using memory above 4GB, a few words might usefully be passed about earlier Windows versions.

As noted above, the PAE kernel dates from Windows 2000. Some sources, including Microsoft's Physical Address Extension, seem to think that the PAE kernel was only for server editions, at least initially. As recently as the January 2007 edition of the MSDN Library on CD, this page of Microsoft's cited just three "systems that can use PAE to take advantage of physical memory beyond 4GB":

- Windows Server 2003, Enterprise Edition
- Windows 2000 Datacenter Server
- Windows 2000 Advanced Server

That was very definitely too small a list, but was surely more about what was intended than what actually was coded and released (if not formally supported). There are two PAE kernels on the installation media for every known release of Windows 2000, Windows XP and Windows Server 2003 (except for Windows 2000 SP2 which seems not to have updated the kernel at all). These each have four kernels:

- NTOSKRNL.EXE, single-processor, without PAE;
- NTKRNLMP.EXE, multi-processor, without PAE;
- NTKRNLPA.EXE, single-processor, with PAE;
- NTKRPAMP.EXE, multi-processor, with PAE.

In practice, Windows is installed with just two kernels, according to whether the machine has one (logical) processor or more. The single-processor kernels have the standard names. The multi-processor kernels are renamed at installation.

As noted above, the kernel in Windows Vista limits memory use by filtering the map of physical memory as received from the loader, so that memory in excess of the limits is discarded before the kernel really starts working with the map. This mechanism dates from at least Windows 2000. At first, there was just the one limit, of total physical memory. A limit on the maximum physical address begins (chronologically) with Windows Server 2003. What's new for Windows Vista is that the limits to memory usage are obtained as data from the registry, using functions whose names make it unarguable

that the discarding of memory in excess of the limits is a licensing issue. In earlier versions than Windows Vista, the memory limits are hard-coded and the connection with licensing is only indirect.

These hard-coded limits are selected according to the product suite, mostly, but sometimes with an extra dependence on something that Microsoft calls 4GT (and is mostly a matter of whether /3GB is given in the BOOT.INI options). The following table of these limits is taken from the single-processor NTKRNLPA.EXE in each release. There may be errors in transcription, there being only so much time that I am willing to take over finding every last detail for this information. If you want more detail or reliability, try finding it from Microsoft.[9]

Version	Package	Total Physical Memory	Maximum Physical Address
5.0.2195.1	Windows 2000	32GB if Datacenter without 4GT; 16GB if Datacenter with 4GT; 8GB if Enterprise; 4GB otherwise	none
5.0.2195.1620	Windows 2000 SP1		
5.0.2195.5438	Windows 2000 SP3		
5.0.2195.6717	Windows 2000 SP4		
5.1.2600.0	Windows XP	64GB if Datacenter without 4GT; 16GB if Datacenter with 4GT; 32GB if Enterprise; 2GB if Blade; 4GB otherwise	none 16GB if Datacenter with 4GT; none if Datacenter without 4GT; none if Enterprise or Blade; 4GB otherwise
5.1.2600.1106	Windows XP SP1		
5.1.2600.2180	Windows XP SP2		
5.1.2600.5512	Windows XP SP3		
5.2.3790.0	Windows Server 2003	128GB if Datacenter without 4GT; 32GB if Enterprise without 4GT; 16GB if Datacenter or Enterprise with 4GT; 2GB if Blade;	16GB if Datacenter or Enterprise with 4GT; 128GB otherwise

		4GB otherwise	
5.2.3790.1830	Windows Server 2003 SP1	2GB if Blade; 128GB if Datacenter without 4GT; 64GB if Enterprise without 4GT; 16GB if Datacenter or Enterprise with 4GT; 4GB otherwise	128GB if Blade; 128GB if Datacenter, Enterprise or Terminal Server without 4GT; 16GB if Datacenter, Enterprise or Terminal Server with 4GT; 4GB otherwise
5.2.3790.3959	Windows Server 2003 SP2		

Note that for nearly 5 years, i.e., from Windows 2000 up to but not including Windows XP SP2, Microsoft did not actually prohibit the use of memory above 4GB by any editions of its current Windows product, whether for servers or clients. Perhaps those incompatible drivers hadn't been noticed yet.

Windows XP SP2

Special mention must be made of Windows XP SP2 and SP3. If you were fortunate enough to have 4GB in a machine for running a client version of Windows up to and including Windows XP SP1, and your hardware had memory remapping so that some of your 4GB was above the 4GB address, and your third-party drivers worked with memory above 4GB, then you will have faced an unfortunate side-effect when upgrading to Windows XP SP2: you will have bought a downgrade of how much RAM Microsoft permits you to use. How well did Microsoft disclose this side-effect in advance?

It was disclosed eventually. The Knowledge Base article The amount of RAM reported by the System Properties dialog box and the System Information tool is less than you expect after you install Windows XP Service Pack 2 reports that enabling PAE in Windows XP SP2 does not allow the use of memory above 4GB. It also talks of changes to the HAL. What actually was changed so strains credibility that I had to be goaded into studying it. As much as anyone talks of 32-bit drivers that simply assume a 32-bit physical address space, it will forever remain that the extreme example of such an assumption is what Microsoft itself coded into the HAL for Windows XP SP2.

Where the Knowledge Base article talks of "unlimited map registers", what it means is that the HAL itself assumes double buffering is never required for any DMA operations on a device that can handle 32-bit physical

addresses.10 In other words, having seen that some 32-bit drivers assume all physical addresses fit 32 bits and thereby think to escape some coding obligations regarding double buffering for their 32-bit device, Microsoft deliberately adopted the same faulty assumption into the HAL, i.e., into arguably the second most critical executable in all of Windows. A company that can do such a thing is clearly not thinking primarily of its product's technical integrity.

Future

It seems unlikely that Microsoft means to remove (or even change) the license limits on memory use by 32-bit Windows Vista any time soon. For the opportunity provided by the first service pack, Microsoft instead decided to play games with how the amount of physical memory is reported. In several places where earlier Windows versions report the amount of RAM that the kernel recognises as usable, Windows Vista SP1 instead reports the total amount of RAM that is installed. For instance, if the original Windows Vista sees only 3069MB of RAM on your machine that has 8GB installed, then the System Properties in Windows Vista SP1 will likely say that you have 8.00GB of RAM. This does not mean, of course, that Windows Vista SP1 actually regards 8GB as usable. RAM that is overridden for hardware support is as lost to Windows Vista SP1 as to the original. RAM in excess of the license limits is discarded by Windows Vista SP1 as by the original. Windows Vista SP1 just doesn't let these losses show as obviously.

Microsoft does acknowledge this "reporting change" in an article Windows Vista SP1 includes reporting of Installed System Memory (RAM). The title is more accurate than might be understood at first glance: reporting the amount of installed system memory is newly coded as an application-level programming feature in the form of a new Windows API function, named GetPhysicallyInstalledSystemMemory, and several programs that are supplied with Windows Vista SP1 use this new feature. Only one, the System Information program (MSINFO32.EXE), is known to contrast the amount of installed RAM with the amount that the operating system believes is usable.

If you're reasonably sharp-eyed, you should be wondering if I can be correct where I say, some distance above, that the kernel discards memory in excess of the license values, such that the memory may as well never have been discovered by the loader from the firmware. If the kernel truly does lose all knowledge of the discarded memory, then how is this memory's existence found by the new function? The answer is of course that the report of installed memory is not obtained directly from the kernel. The kernel's only involvement is in fetching the raw SMBIOS firmware table, which is

then interpreted at application level to discover the amount of installed memory. The SMBIOS table tells such things as that the system board has 8 memory chips at 1GB each. It doesn't tell anything about how (or if) that RAM is addressable, let alone how much of it is usable by Windows.

When running on earlier Windows versions, programs that want to discover the amount of installed RAM can do their own interpretation of the SMBIOS table, having got it by calling GetSystemFirmwareTable. When running on much earlier Windows versions, there is perhaps no better way than to query Windows Management Instrumentation (WMI) for its knowledge of the SMBIOS table, which is hardly a trivial exercise.[11] That such mucking around for one seemingly simple piece of data is now made as easy as calling one API function is very welcome. It's sadly typical that Microsoft didn't implement it as a programming feature of its operating system until Microsoft needed it for Microsoft's purposes. It's a pity that those purposes in this particular case have an element of disguise.

Of course, there's a very good argument that the amount of RAM shown in the System Properties ought all along to have been the amount installed. But it hasn't been all along, and to change it now, just as increasingly many users see the amount shown and wonder why it is so much less than the amount they know to be installed, looks like sharp practice.

Microsoft finally attended to this in Windows 7, which has the System Properties show the amount installed but follow it with the backwards-compatible report of what's usable.

Windows Server 2008

Perhaps you think that because Microsoft is the only true authority on its own software, there remains some scope for technical issues to explain the prevention of 32-bit Windows Vista from using memory above 4GB. After all, Microsoft's kernel-mode programmers are surely among the best and brightest and they have access to the source code. How can anyone outside Microsoft be truly certain what code is in Windows? For instance, could it be that although the kernel for 32-bit Windows Vista has some code for using memory above 4GB, it does not have the complete code? Or could it be that the code in 32-bit Windows Vista was merely in transition from Windows Server 2003 to Windows Server 2008 and does not work entirely correctly? Only Microsoft can know.

To some extent, you would be right, but then I say to you: consider Windows Server 2008. For the loader and kernel in Windows Vista SP1 (and, by the way, for the overwhelming majority of all executables), the

corresponding executable in Windows Server 2008 is exactly the same, byte for byte. Yet Microsoft sells 32-bit Windows Server 2008 for use with as much as 64GB of memory. Does Microsoft really mean to say that when it re-badges these same executables as Windows Vista SP1, they suddenly acquire an architectural limit of 4GB? Or is it that a driver for Windows Server 2008 is safe for using with memory above 4GB as long as you don't let it interact with the identical executables from Windows Vista SP1?

Summary

Microsoft has always limited the amount of memory that Windows will use. Different Windows products have different limits. For 32-bit Windows Vista, the limit of 4GB is unarguably coded as a licensing issue but Microsoft does not say this explicitly in the supposed License Agreement (or anywhere else).

This has not been a problem for consumers until recently because the license limit of 4GB for client editions of Windows, such as Windows XP and Windows Vista, permitted the use of far more memory than many consumers ever thought to dream about. Even now that machines with 4GB of RAM are no rarity, many consumers (including me) typically don't run enough memory-hungry programs to be using more than one or two gigabytes in total. Yet clearly the day is not far off that 4GB will be a typical installation. Software will inevitably grow to use what's there. Soon, machines with more than 4GB will not be regarded as monsters.

A perception has developed, and even become widespread, that exceeding 4GB requires a mass migration to 64-bit Windows. At best, this perception misunderstands both the design of Intel's processors and a decade-old development in 32-bit Windows. Although Microsoft itself seems never to say explicitly that 4GB is a barrier for 32-bit Windows in general, Microsoft does say that 32-bit Windows Vista in particular is incapable of using 4GB and that the workaround necessarily includes using 64-bit Windows Vista instead.

The mechanism by which 32-bit Windows Vista is incapable of using memory above 4GB is simply that Microsoft does not license it to use memory above 4GB. The code to use memory above 4GB is already present in the product as shipped. Microsoft just doesn't license you to use it. Microsoft seems never to say explicitly that this is the mechanism.

Microsoft does say that memory use by 32-bit Windows Vista is limited "to avoid potential driver compatibility issues" but the arguments are weak,

especially for new computers with new hardware and new drivers. Moreover, Microsoft does not open its arguments to independent testing, given that the license values are protected from being tampered with. Even for machines on which the incompatibilities are real, they would be avoidable through configurable options that exist anyway. That they must be avoided (or even are better avoided) through the licensing mechanism is again not something that Microsoft seems to have explained anywhere.

A suspicion seems reasonable that Microsoft is at least content to have the mass market perceive 4GB as an architectural limit to 32-bit Windows, so that the inevitable increase in the amount of RAM fitted to a typical computer will itself move the consumer base to 64-bit Windows and thence to the 64-bit applications that users will quickly see as natural purchases for their 64-bit operating system. There are, of course, very good reasons to migrate to 64-bit Windows, but the change should be assessed on its merits, not taken as necessary because staying with 32-bit Windows is closed off as an option. Imposing an artificial limit at an amount of memory that conveniently coincides with a widely believed myth has the look of a marketing ruse to bring forward the migration to 64 bits without fully-informed assessment. Microsoft certainly has not informed, and the rest of the industry has not questioned how and why 32-bit Windows Vista is constrained to ignore memory above 4GB. This is an abuse that consumers should not have to tolerate. Someone with authority over Microsoft ought investigate whether Microsoft's descriptions of 32-bit Windows Vista as being incapable of using memory above 4GB are misleading or illegal.

Thanks[i]

[i] Geoff Chappell - Software Analyst
A Prince NRVL presentation.