How exactly are page tables allocated on demand for large reserved regions?

devblogs.microsoft.com/oldnewthing/20160701-00

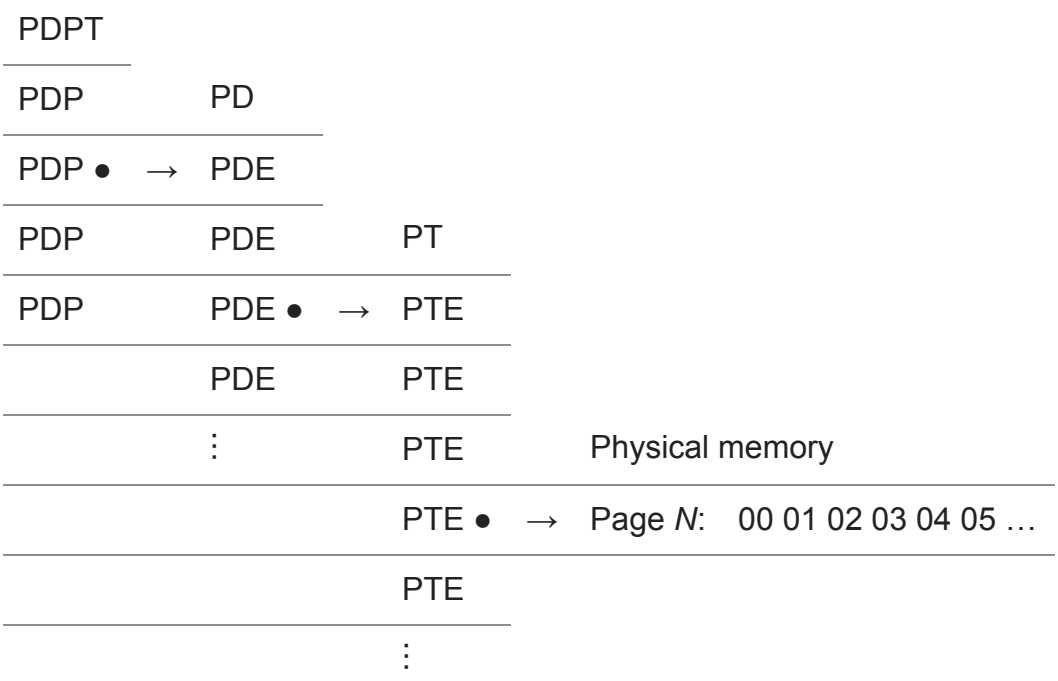
July 1, 2016

Raymond Chen

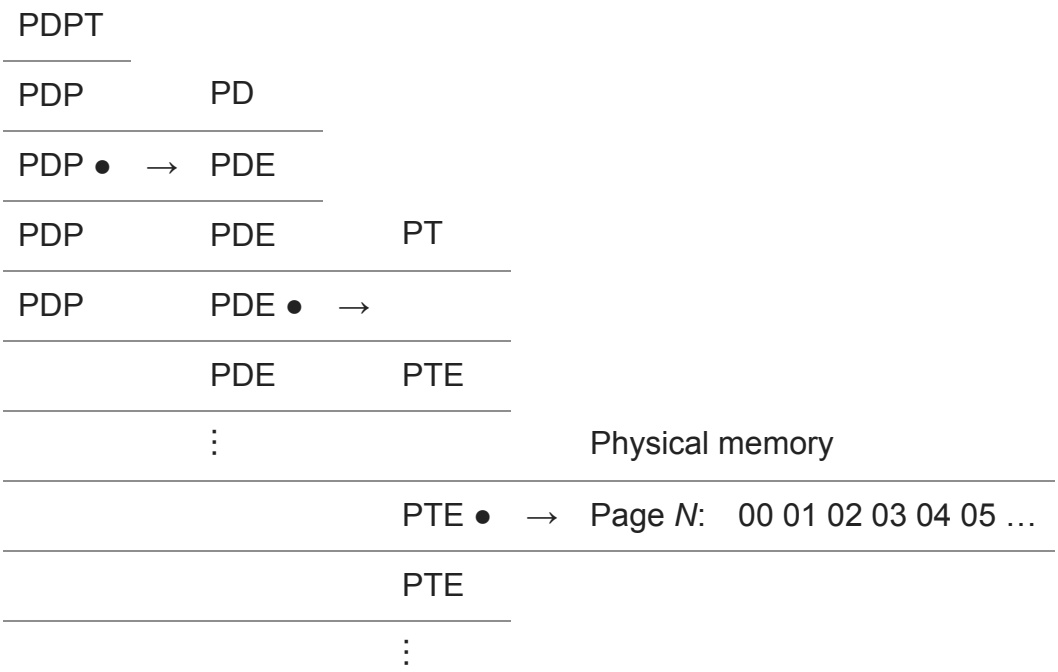
Commenter pm100 asked to go into more detail into why reserving a large amount of address space used to be expensive, and now it's not.

I sort of answered it in my reply, but let's draw some pictures.

Recall that the paging structure is hierarchical. To describe a page of memory, you first select a page directory pointer from the page directory pointer table (PDPT). That points to a page directory (PD). You then select a then a page directory entry (PDE) from the page directory. This points to a page table (PT). You then select a page table entry (PTE) from the page table. The page table entry tells you where the memory for the page can be found.

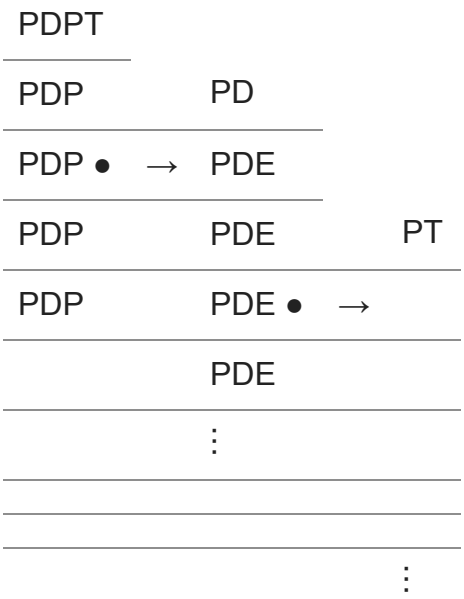


For pages that are reserved, the page table entry doesn't point to physical memory. Instead, it has a sad-face that says "If you try to access this memory, you will get a page fault."



The above diagram shows a page table with some pages reserved (the sad faces), and some pages committed and present (the ones with a filled-in PTE).

If you reserved a huge chunk of address space, the traditional way of representing this was a page table full of sad-faces.



But starting in Windows 8.1 and Windows Server 2012 R2, the memory manager optimizes this out, and instead of creating an entire page table filled with sad faces, it puts a sad face *in the page directory*:

PDPT		
PDP		PD
PDP ●	→	PDE
PDP		PDE
PDP		●
		PDE
		⋮

If you try to access memory through a sad-face page directory entry, the CPU raises a page fault.

Looking back at the original issue: The customer was reserving 100GB of address space. Each page table maps 512 × 4KB = 2MB of address space, so that's 51,200 page tables filled with sad faces. The memory manager optimizes this out and instead of creating 51,200 page tables filled with sad faces, it creates 51,200 page directory entries with sad faces.

But wait, that's still proportional to the size of the memory reserve, albeit with a much lower constant factor. How do we get the memory usage to be constant?

Page directory entries are grouped together into page directories. Each page directory holds 512 page directory entries, and if the page directory is filled with sad faces, we can replace it with a sad face in the page directory pointer table. Our 51,200 sad-face page directory entries become 100 sad-face page directory pointer. (On x64, there's another hierarchy level above the page directory pointer table, known as the page map level 4, but let's ignore that for now.)

A reservation of 100GB of address space turns into this:

PDPT
PDP
⋮
PDP
PDP

Okay, so the constant factor is now a lot lower. And if you filled the entire page directory page table with sad faces, then it can be replaced by a single sad face in the page map level 4.

So now we have an extremely low constant factor, but it's still a constant factor. How do you get the whole thing to be a constant?

Because eventually you will hit the top of the virtual memory hierarchy (currently page map level 4), and the madness stops.

If you think about it another way, you would have realized that reserving address space would never have required creating page tables, page directories, or page directory pointer tables in the first place: The address space was already invalid (namely, in the free state). Reserving address space doesn't make the pages valid; they're just invalid for a different reason (namely, in the reserved state). They were sad faces before, and they remain sad faces. Either way, they're still sad. You didn't have to do anything with the page tables to change sad faces to sad faces.

There is therefore no additional cost in terms of page tables, page directories, or page directory pointer tables. The cost is in the memory manager's internal bookkeeping, which is constant.

Raymond Chen

Follow