Page Tables

From OSDev Wiki

The page tables (or page map levels) are used to map each virtual page to a corresponding physical page. Zero or more virtual pages can correspond to the same physical page. The size of a page depends on the processor mode (protected, compatibility or long mode), the extensions used (e.g. PAE) and the virtual address bits supported by the processor (current AMD64 processors support up to 48-bit virtual addresses).

Contents

Determining your page size

To determine the ideal page size, you can't just look at the overhead of the paging structures alone.

Typically (for user space), each process has at least 3 areas with different characteristics: executable, read only no-execute and read/write no-execute. When paging is used to enforce this, you end up with padding from the actual end of each area up to the next page boundary. You can assume that on average this padding will be 50% of the page size. For example, with 4096 byte pages and 3 areas you'd expect 6 KiB of RAM wasted per process due to padding, and with 2 MiB pages you'd expect 3 MiB of RAM wasted per process.

Typically (for OSs like Windows and Linux) there's around 50 processes where most use very little RAM and some use lots (X, browser, etc). If you assume 50 processes that use an average of 10 MiB of RAM each; then (for 4 KiB pages in long mode) each process (on average) would use a PML4, PDPT, PD and 5 page tables; or 32 KiB for all paging structures. For 2 MiB pages each process (on average) would use a PML4, PDPT and a PD; or 12 KiB for all paging structures.

That gives us some rough figures for comparison. For 50 processes where each process is an average of 10 MiB and has 3 different areas:

- **4 KiB paging will cost about 6 KiB for padding and 32 KiB for paging structures, or about 38 KiB of overhead per process**
- **2 MiB paging will cost about 3 MiB for padding and 12 KiB for paging structures, or about 3084 KiB of overhead per process**
- **4 KiB paging will cost about 1.86 MiB of overhead for all 50 processes combined**
- **2 MiB paging will cost about 150.6 MiB of overhead for all 50 processes combined**
- **With 500 MiB of RAM actually used by all processes; 1.86 MiB of total overhead works out to about 0.37% more, which is almost nothing**
- **With 500 MiB of RAM actually used by all processes; 150.6 MiB of total overhead works out to about 23.15% more, which is massive**

For performance (e.g. TLB misses) it's much harder to estimate the likely cost, as it depends on how much of the paging structures remain in the cache (and how much has to be fetched from RAM), how large the TLBs are (for both small pages and large pages), if the CPU caches higher level structures (modern CPUs do), the working set of each process and its access pattern, how often you switch between processes, etc...

However (for typical OSs with typical loads), I'd assume it's very unlikely that the performance gains you get from using 2 MiB pages is going to justify having roughly 23% more RAM wasted.

Basically, 4 KiB pages (with 4 levels of paging structures) is starting to get a little small, but the next step up (2 MiB pages with 3 levels of paging structures) is far too big to be practical for most things.

To reduce the number of levels of paging structures, a better idea would be to also increase the size of page directories, PDPTs, etc. For example, for 55-bit virtual addresses you could have 64 KiB pages, 64 KiB page tables, 64 KiB page directories and 64 KiB PDPTs. Unfortunately we have to wait for Intel (or AMD) to do something like that though.

~ Brendan (http://forum.osdev.org/viewtopic.php?p=198045#p198045)

Luckily though you can freely mix 4k, 2MiB and 1GiB pages. You don't have to use an uniform page size for everything. So a process having 9MB of data can get 4 2MiB pages and make up the rest with 4k pages. That saves paging structures, improves TLB usage and does not increase the overhead at all.

Recursive mapping

To make it easier to change the current address space's page map, you can map an entry of the highest page map level into itself.

Recursive mapping wastes some virtual address space. This table shows the relative space used by the recursively mapped page table:

Mode	Page size	Max page map size	Used virtual space	Total virtual space	Ratio
Protected mode (non-PAE)	4 KiB	4 MiB	4 MiB	4 GiB	1/1024 (0.1%)
	4 MiB	4 KiB			
Protected mode (PAE)	4 KiB	8 MiB	1 GiB	4 GiB	1/4 (25%)
	2 MiB	16 KiB			
Long mode (48-bit)	4 KiB	512 GiB	512 GiB	256 TiB	1/512 (0.2%)
	2 MiB	1 GiB			
	1 GiB	2 MiB			

The *Recursive mapping* column in the tables below shows the base address and offset to use to get to a particular page map level when the page map is recursively mapped as the *last* entry.

Protected/compatibility mode (32-bit) page map

In protected mode, the virtual address space is 32-bit (4 GiB) in size, regardless of whether PAE is enabled or not.

Non-PAE mode

Without enabling PAE, the page map can reference physical pages with addresses up to 32-bit (4 GiB).

4 KiB pages

Level	Table	Size	Range	Bits	Entries	Pages	Recursive mapping
0	(page)	-	0x1000 (4 KiB)	12 bits	-	0x1 (1)	-
1	PT	0x1000 (4 KiB)	0x40 0000 (4 MiB)	10 bits	1024	0x400 (1024)	0xFFC0 0000 + 0x1000 * PDi
2	PD	0x1000 (4 KiB)	0x10000 0000 (4 GiB)	10 bits	1024	0x10 0000 (1048576)	0xFFFF F000

4 MiB pages

Level	Table	Size	Range	Bits	Entries	Pages	Recursive mapping
0	(page)	-	0x40 0000 (4 MiB)	22 bits	-	0x1 (1)	-
2	PD	0x1000 (4 KiB)	0x10000 0000 (4 GiB)	10 bits	1024	0x400 (1024)	0xFFC0 0000

PAE mode

With PAE, the page map can reference physical pages with addresses up to 36-bit (64 GiB).

4 KiB pages

Level	Table	Size	Range	Bits	Entries	Pages	Recursive mapping
0	(page)	-	0x1000 (4 KiB)	12 bits	-	0x1 (1)	-
1	PT	0x1000 (4 KiB)	0x20 0000 (2 MiB)	9 bits	512	0x200 (512)	0xC000 0000 + 0x20 0000 * PDi + 0x1000 * PTi
2	PD	0x1000 (4 KiB)	0x4000 0000 (1 GiB)	9 bits	512	0x40000 (262144)	0xC060 0000 + 0x1000 * PDi
3	PDP	0x20 (32 bytes)	0x10000 0000 (4 GiB)	2 bits	4	0x10 0000 (1048576)	0xC060 3000

2 MiB pages

Level	Table	Size	Range	Bits	Entries	Pages	Recursive mapping
0	(page)	-	0x20 0000 (2 MiB)	21 bits	-	0x1 (1)	-
2	PD	0x1000 (4 KiB)	0x4000 0000 (1 GiB)	9 bits	512	0x200 (512)	0xC000 0000 + 0x20 0000 * PDi
3	PDP	0x20 (32 bytes)	0x10000 0000 (4 GiB)	2 bits	4	0x800 (2048)	0xC060 0000

Long mode (64-bit) page map

In long mode, the virtual address space *could in theory* be 64-bit (16 EiB) in size, but individual processors allow only a portion of that space to be addressed. The currently most common processor implementations allow a 48-bit (256 TiB) address space. For such virtual addresses bits 48-63 must be a copy of bit 47 (similar to sign extension), and this splits the virtual address space in a higher half and a lower half.

48-bit virtual address space

4 KiB pages

Level	Table	Size	Range	Bits	Entries	Pages	Recursive mapping
0	(page)	-	0x1000 (4 KiB)	12 bits	-	0x1 (1)	-
1	PT	0x1000 (4 KiB)	0x20 0000 (2 MiB)	9 bits	512	0x200 (512)	0xFFFF FF80 0000 0000 + 0x4000 0000 * PDPi + 0x20 0000 * PDi + 0x1000 * PTi
2	PD	0x1000 (4 KiB)	0x4000 0000 (1 GiB)	9 bits	512	0x40000 (262144)	0xFFFF FFFF C000 0000 + 0x20 0000 * PDPi + 0x1000 * PDi
3	PDP	0x1000 (4 KiB)	0x80 0000 0000 (512 GiB)	9 bits	512	0x800 0000 (134217728)	0xFFFF FFFF FFE0 0000 + 0x1000 * PDPi
4	PML4	0x1000 (4 KiB)	0x10000 0000 0000 (256 TiB)	9 bits	512	0x10 0000 0000 (68719476736)	0xFFFF FFFF FFFF F000

2 MiB pages

Level	Table	Size	Range	Bits	Entries	Pages	Recursive mapping
0	(page)	-	0x20 (2 0000 MiB)	21 bits	-	0x1 (1)	-
2	PD	0x1000 (4 KiB)	0x4000 (1 0000 GiB)	9 bits	512	0x200 (512)	0xFFFF FF80 0000 0000 + 0x4000 0000 * PDPi + 0x20 0000 * PDi
3	PDP	0x1000 (4 KiB)	0x80 0000 0000 (512 GiB)	9 bits	512	0x40000 (262144)	0xFFFF FFFF C000 0000 + 0x20 0000 * PDPi
4	PML4	0x1000 (4 KiB)	0x10000 0000 0000 (256 TiB)	9 bits	512	0x8000000 (134217728)	0xFFFF FFFF FFE0 0000

1 GiB pages

Level	Table	Size	Range	Bits	Entries	Pages	Recursive mapping
0	(page)	-	0x4000 (1 0000 GiB)	30 bits	-	0x1 (1)	-
3	PDP	0x1000 (4 KiB)	0x80 0000 (512 0000 GiB)	9 bits	512	0x200 (512)	0xFFFF FF80 0000 0000 + 0x4000 0000 * 0000 PDPi
4	PML4	0x1000 (4 KiB)	0x10000 (256 0000 0000 TiB)	9 bits	512	0x40 000 (262144)	0xFFFF FFFF C000 0000

See Also

Articles

- Paging
- Page Frame Allocation

Forum

- **Time for 2 MB pages? (http://forum.osdev.org/viewtopic.php?f=1&t=24292)**

Retrieved from "http://wiki.osdev.org/index.php?title=Page_Tables&oldid=17790"
Category: Memory management
