How does multi-level page table save memory space?

I am trying to understand how multi-level page table saves memory. As per my understanding, Multi-level page table in total consumes more memory than single-level page table.

Example : Consider a memory system with page size 64KB and 32-bit processor. Each entry in the page table is 4 Bytes.

Single-level Page Table : 16 (2^16 = 64KB) bits are required to represent page offset. So rest 16-bits are used to index into page table. So

Size of page table = 2^16(# of pages) * 4 Bytes(Size of each page table entry) = 2^18 Bytes

Multi-level Page Table : In case of two-level page table, lets use first 10-most significant bits to index into first level page table. Next 10-bits to index into second level page table, which has the page number to frame number mappings. Rest 12-bits represents the page offset.

Size of a second-level page table = 2^10 (# of entries) * 4 bytes(size of each entry) = 4 KB

Total size of all the second-level page tables = 2^10 (# of second-level page tables) * 4KB (Size of each second-level page table) = 4 MB

Size of first-level page table = 2^10(# of entries) * (10/8) Bytes (Size of each entry) = 1.25 KB

Total memory required to store first and second level page tables = 4 MB + 1.25 KB

So we need more memory to store multi-level page tables.

If this is the case, How does multi-level page tables save memory space ?

memory operating-system paging virtual-memory page-tables

asked Apr 6 '15 at 7:58

Anil Kumar K K
192 2 12

All page table entries don't have to be present in memory at the same time. Just the top-level dictionary, the rest can be swapped out to the disk and loaded and used when needed (if ever). So the saving is (in my opinion) in the fact that the page map occupies modest amount of memory – xmojmr Apr 6 '15 at 8:16

3 Answers

1. In singlelevel pagetable you need the whole table to access even a small amount of data(less memory references). i.e 2^20 pages each PTE occupying 4bytes as you assumed.

Space required to access any data is 2^20 * 4bytes = 4MB

2. Paging pages is multi-level paging.In multilevel paging it is more specific, you can with the help of multi-level organization decide which specific page among the 2^20 pages your data exists, and select it . So here you need only that specific page to be in the memory while you run the process.

In the 2 level case that you discussed you need 1st level pagetable and then 1 of the 2^10 pagetables in second level. So, 1st level size = 2^10 * 4bytes = 4KB 2nd level we need only 1 among the 2^10 pagetables = so size is 2^10 * 4bytes = 4KB

Total size required is now : 4KB + 4KB = 8KB.

Final comparision is 4MB vs 8KB .

answered May 27 '15 at 1:36

Sai
86 4

2 Well explained. I was wondering what will be the content of page table at level 2. I understand, in level 1 the contents are frame number and some bits (dirty, modified, etc). – Deepak Dec 10 '15 at 11:15

Here is a primary advantage of multilevel page tables:

> First, chop up the page table into page-sized units; then, if an entire page of page-table entries (PTEs) is invalid, don't allocate that page of the page table at all.

Source. (Section 20.3)

Thus the amount of memory needed for the page table is not dictated by the size of the address space, but by the amount of memory that the process is using.

In addition, the page of page table entries can itself be paged if physical memory gets full - only the page directory needs be always present in memory.

answered Apr 6 '15 at 8:29

Craig S. Anderson
3,594 3 15 31

1 I was wondering what will be the content of page table at level 2. I understand, in level 1 the contents are frame number and some bits (dirty, modified, etc). – Deepak Dec 10 '15 at 11:15

Multi-level tables are primarily needed because if the memory structure in Intel-land.

Let's suppose you have a 32-bit system and you divide the address space so that the top have is reserved to the system and the lower half is for user addresses.

With such a division, you would need 2GB worth of contiguous page table entries in every user page table to to reach the system addresses.

The old VAX to a simple approach to this. It divided the 4GB address space into 4 regions (2 user, 1 system, one unusable). The three usable areas had its own page table.

Each regions had its own page table. Because there was a dedicated system address space, the user page tables could be virtual addresses so they would not require contiguous memory.

The first phase of address translation was to look at the 2 high order address bits to select the page table to use.

Instead of having separate page tables, Intel-land breaks the page table up. That lessens the problems of (1) needing contiguous memory for the table; (2) requiring the page table to span the full address space; (3) allows the definition of a kernel addresses that can be shared by all processes.

answered Apr 10 '15 at 18:46

user3344003
7,880 1 8 26