Problems

1. Collection of problems from review session, along with a few new ones
2. Problems I consider harder marked (*)
3. Unit reference (using binary units to be precise, if you don't understand this ignore the 'i'):
   (a) 1 KiB = 2^{10} bytes 10^3 bytes
   (b) 1 MiB = 2^{20} bytes 10^6 bytes
   (c) 1 GiB = 2^{30} bytes 10^9 bytes
   (d) 1 TiB = 2^{40} bytes 10^{12} bytes
   (e) 1 PiB = 2^{50} bytes 10^{15} bytes
4. Warmup: Given the following number of bits in a virtual address, physical address, and pagesize, calculate the following values:
   (a) VA: 56 bits, PA: 48 bits, pagesize: 16 KiB
   (b) VPN bits
   (c) PPN bits
   (d) Offset bits
   (e) How many virtual pages possible?
   (f) How many physical pages possible?
5. X86-64 pagetable rapid-fire from 2.2 (Multi-indexed):
   (a) How many entries are there in an L1 pagetable?
   (b) How many possible pages are accessible from a given L2 pagetable?
   (c) How many possible L3 pagetables are there if virtual address space fully used?
   (d) How many memory accesses are needed to access a given memory address, assuming no faults or errors?
   (e) (*) If each pagetable entry uses 8 bytes, why is a 9-bit index extra nice?
6. Page allocation from 2.2 (Multi-indexed):
   (a) How many pages in the x86-64 architecture needed to allocate just one byte of memory? [draw pagetable tree]
   (b) How many pages needed to allocate 5 pages worth of memory? [extend tree by appending 4 more pages]
   (c) 2^9? (both best and worst-case) [prompt for both optimal and pessimal cases, ask for what the pagetable tree looks like in each case]

(d) $2^9 + 1$? (in best case)

(e) What if I spawned 32 processes, each allocates 8 pages, in best case? [32 * (8+4) = 2^7 * 3 = 384]

(f) (*) Finally, 2 processes, one which allocates 1 GiB (2^{30}), another which allocates 1 KiB (2^{10}) (again, best case)

7. Memory reads from 2.2 (Multi-indexed):

   (a) Greater number of memory queries required per address if we solely translate using page tables

      i. If no caching, how many memory accesses needed to execute a simple instruction?

      ii. 0x500 movq 0x200000, %rax

   (b) TLB solves this by caching the page translations that are used the most

      i. If both previous page translations cached, but not actual memory cached, how many memory accesses needed to then exec?

      ii. 0x504 movq 0x200008, %rbx

8. (*) Huge Pages from 4. (Huge Pages):

   (a) If an entry on an L3 page table pointed straight to a huge page, what size would make the most sense for this huge page?

   (b) How many such pages could we allocate at once?

   (c) If we wanted to allocate 2 huge pages and 2000 regular pages, what is the minimum number of 4 KiB pages needed to allocate? (treat allocating 1 huge page as allocating the same amount of memory's worth in regular pages)

   (d) We can go even further beyond, can define a 1 GiB (2^{30} byte) "gigantic" page

   (e) What level pagetable makes the most sense for entries that point to these gigantic pages?

9. x86 Variants

   (a) In the context of x86-64 architecture, if each pagetable entry takes 8 bytes, how many bytes would be needed to store a complete linear pagetable? (binary and human form) How does this compare to modern-day computer capabilities?

   (b) The x86 32-bit architecture, obviously, uses 32 bits for virtual addressing

      i. Instead of 4 pagetable layers, it has 2, while pages remain at 4 KiB

      ii. How many bits are given to each pagetable index?

iii. How many entries would each pagetable have?

iv. How large is each pagetable, assuming each entry takes 4 bytes?

v. (*) In each pagetable entry, how many bits are unused by the PPN?

(c) (*) Imagine we changed the x86-64 architecture a little

i. Instead of 4 PTE layers, we set up 3

ii. Uneven, L1 index takes 18 bits, L2 and L3 each take 9 bits

iii. How large would the L1 pagetable be? (binary and human)

iv. If I wanted to allocate 1 page under this regime, how many physical pages needed?

v. 513? (best case)

Answers

1. Warmup
   (a) 56 - $\log_2(16 * 1024) = 56 - 14 = 42bits$
   (a) 48 - 14 = 34 bits
   (b) $\log_2(16 * 1024) = 14bits$
   (b) 2^{42} = 4 trillion
   (c) 2^{34} = 16 billion
2. Rapid-fire
   (a) 512, 2^9 indices for an L1 pagetable
   (b) 2^{27} = 128 million, L2 can reach 2^9 L3, each of which can reach 2^9 L3, each of which can reach 2^9 L4 so $2^{(9+9+9)}$
   (c) 2^{18} = 256 thousand, L1 can reach 2^9 L2, each of which can reach 2^9 L3, so $2^{(9+9)}$
   (d) 5, walk L1-L4, then read actual page
   (e) (*) There are 512 entries per pagetable, so 8 bytes each means that a pagetable takes 2^12 bytes, which is a page
3. Page allocation
   (a) 5 (L1 - L2 - L3 - L4 - actual page)
   (b) 9 (L1 - L2 - L3 - L4 - actual pages)
   (c) 2^9 + 4 in best case (ditto), 4 * 2^9 + 1 = 2^{11} + 1 = 2049 in worst case (each page requires own L2, L3, L4)
   (d) 2^9 + 6 (second L4 page table needed)
   (e) 32 * (8+4) = 2^7 * 3 = 384, each process needs 12 pages to allocate 8 pages
   (f) (*) 2^{18} + 2^9 + 8 (L1 - L2 - L3 - 2^9 L4 - 2^{18} pages + 5 pages for small)
4. Memory reads
   (a) 2 * 5, each memory access requires full walk of pagetables
   (b) 6, TLB caches page translation for latter read, not for former
5. (*) Huge Pages
   (a) 2^{21}, take L4 index bits transfer to offset
   (b) 2^{27} (all 2^{27} PTEs in all possible L3 pagetables point to a huge page)
   (c) 3 + 2 * 512 + 4 + 2000 = 3037 (L1 - L2 - L3 - 2 huge pages and 4 L4 which point to regular pages)

(d) L2, by same logic as first question, a 2^{30} byte page needs 2^{30} offset bits, so take L3 and L4 indices, so L2 points directly to gigantic

6. x86 Mutations

   (a) x86 32-bit

      i. 2^{36} virtual pages * 2^3 bytes per PTE = 2^{39} bytes = 512 GB, which vastly outstrips personal computers
      ii. 10 bits (20 VPN bits / 2)
      iii. 2^{10} = 1024 PTEs
      iv. 4096 bytes (1024 PTEs * 4 bytes / PTE)
      v. (*) 12 bits (32 bits in PTE
      vi. 20 bits for PPN)

   (b) Collapsed indices

      i. 2^{21} bytes = 2 million bytes (2^{18} PTEs * 2^3 bytes per entry)
      ii. 515 (512 pages for L1 + 1 L2 + 1 L3 + 1 actual page)
      iii. 1028 (512 pages for L1 + 1 L2 + 2 L3 + 513 actual pages)