CSCI 3150 Tutorial 3 - Virtual Memory

XUE Jin (jinxue@cse.cuhk.edu.hk)

November 3, 2022

Agenda

- Some sample questions related to virtual memory for the assignment and exams
  - Calculation: given the configuration of virtual memory, calculate some parameters
  - Address translation: given the page mapping and some virtual addresses (VA), find out the corresponding physical addresses (PA)
  - Page table setup: given the relationship between PA and VA, set up the page tables that represents the mapping

Sample Question - Calculation

- Suppose the page size is 8 bytes, the first-level page table has 4 entries and the second-level page table has 8 entries.
    1. What is the size of the virtual address space?
    2. How many bits does a virtual address have?
    3. How many bits should be reserved for the first-level page table index, the second-level page table index and the offset respectively?

First-level index (__ bits)	Second-level index (__ bits)	Offset (__ bits)

Solution

1. What is the size of the virtual address space?
   - The 2^{nd} -level page table has 8 entries (pages of 8 bytes) $\Rightarrow$ one 2^{nd} -level page table maps to $8 \times 8 = 64$ bytes
   - The 1^{st} -level page table has 4 entries (referencing 2^{nd} -level page tables) $\Rightarrow$ one 1^{st} -level page table maps to $4 \times 64 = 256$ bytes
   - The 1^{st} -level page table is also the top level, thus the virtual address space has 256 bytes
2. How many bits does a virtual address have?
   - The virtual address space has 256 bytes $\Rightarrow$ we have 256 unique VAs
   - $256 = 2^8$
   - Thus we need 8 bits for a VA

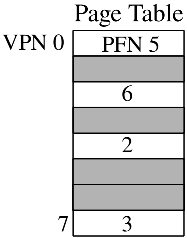
Solution (Cont'd)

1. How many bits should be reserved for the first-level page table index, the second-level page table index and the offset respectively?
   - A page has 8 bytes (8 unique offsets)
     - $8 = 2^3$
     - Need 3 bits to locate an offset inside a page
   - A 2^{nd} -level page table has 8 entries
     - $8 = 2^3$
     - Need 3 bits to locate an entry inside a 2^{nd} -level page table
   - For the same reason, we need 2 bits to locate an entry inside a 1^{st} -level page table
   - Also notice that $3 + 3 + 2 = 8$ is exactly the length of a VA

First-level index (__ bits)	Second-level index (__ bits)	Offset (__ bits)

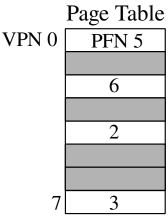
Sample Question - Address Translation

- Suppose page size is 8 bytes. Given the following page table:
  1. Which of the virtual addresses are mapped?
     - 19, 45
  2. What are the corresponding physical addresses for them if mapped?
- Numbers in entries are phyiscal page frame number (PFN)
- Entries in gray color are not mapped



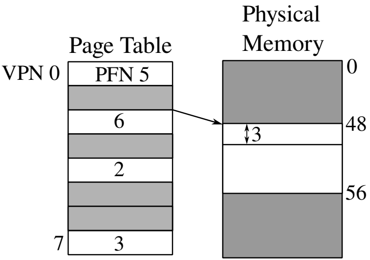
Solution

1. For VA=19
   - The virtual page number (VPN) is $floor(19/8) = 2$
   - You can also use bit-shift to calculate the VPN
     - Page size $= 8 \Rightarrow$ offset has 3 ($log_2 8$) bits
     - VPN can be calculated by skipping (right-shift) all offset bits $19 >> 3 = 2$
   - $PT[2] = 6$ so VPN 2 is mapped to PFN 6
2. For VA=45
   - VPN is $floor(45/8) = 5$
   - $PT[5]$ is not mapped



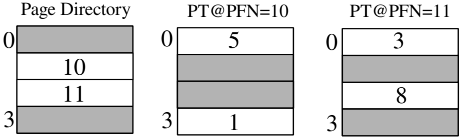
Solution (Cont'd)

- We know that only VA=19 is mapped and it is mapped to PFN 6
  - The physical base address of the page is $6 \times 8 = 48$
- The offset of VA=19 within the virtual page is $19\%8 = 3$
- By adding the offset to the physical base address we can get the PA for VA=19
  - PA $= 48 + 3 = 51$



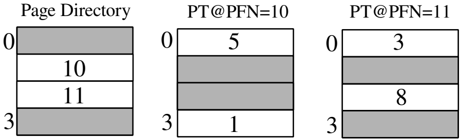
Sample Question - Two-Level Address Translation

- Page size is still 8 bytes but we have two levels (a page directory and some page tables)
- Translate the following VAs into PAs:
  - 34, 75, 100



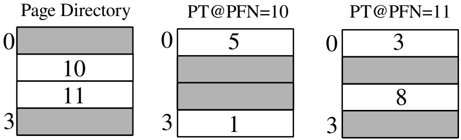
Solution

- A page size is 8 bytes and a PT has 4 entries $\Rightarrow$ a PT maps $4 \times 8 = 32$ bytes
- For VA=34
  - First, we need to find out which PT maps this address
  - VPN1 $= floor(34/32) = 1$ so it's the second PT (page directory entry 1)
  - Using the page directory, we know that entry 1 is PT@PFN=10
  - The offset within that PT is $34\%32 = 2$
  - We can continue to PT@PFN=10 and find out the corresponding PA for the byte at offset 2 inside the PT



Solution (Cont'd)

- In PT@PFN=10, we want to find the PA for the byte at offset 2
- VPN2=$floor(2/8) = 0$ so it's the first page in the PT (PFN=5)
- The offset inside that page is $2\%8 = 2$
- So the PA for VA=34 is $5 \times 8 + 2 = 42$



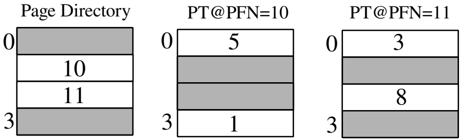
Solution (Cont'd)

- Another way is to use the number of bits for each part that we calculate previously
- VPN1: 2 bits (4 entries), VPN2: 2 bits (4 entries), offset: 3 bits (8 bytes page size)
- The binary representation of 34 is 0100010_b
- Thus we can directly obtain VPN1 $= 01_b = 1$, VPN2 $= 00_b = 0$, Offset $= 010_b = 2$
  - Page directory entry 1, page table entry 0 and page offset 2

VPN1 (2 bits)		VPN2 (2 bits)		Offset (3 bits)		
0	1	0	0	0	1	0

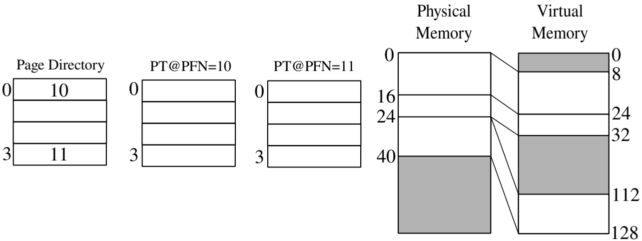
Solution (Cont'd)

- For VA=75
  - VPN1=*floor*(75/32) = 2, page directory entry 2 is PT@PFN=11
  - Offset within that PT is 75%32 = 11
  - VPN2=*floor*(11/8) = 1, entry 1 in PT@PFN=11 is not mapped
  - Thus VA=75 is not mapped
- For VA=100
  - VPN1=*floor*(100/32) = 3, page directory entry 3 is not mapped
  - Thus VA=100 is not mapped



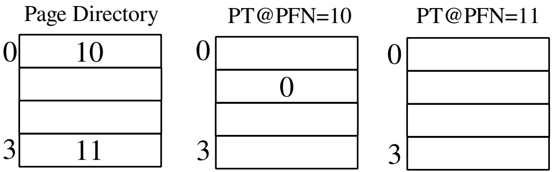
Sample Question - Page Mapping Setup

- A program has three pieces of data that should be loaded and mapped
  1. Code segment is loaded at physical memory range [0, 16) and should be mapped to virtual memory range [8, 24)
  2. Data segment: PA [16, 24) and should be mapped to VA [24, 32)
  3. Stack segment: PA [24, 40) and should be mapped to VA [112, 128)
- You should fill in the following page tables to set up the page mapping



Solution

- The code segment has 2 pages: PA $[0, 8)$ (PFN=0) $\rightarrow$ VA $[8, 16)$ and PA $[8, 16)$ (PFN=1) $\rightarrow$ VA $[16, 24)$
- For the first page, virtual base address is 8
  - VPN1 $=$ $floor(8/32) = 0$, PT offset is $8\%32 = 8$
  - VPN2 $=$ $floor(8/8) = 1$
- So it is the first page directory entry (PT@PFN=10) and the second page inside that PT
- We can fill the PFN in that entry



Solution (Cont'd)

- Repeat this process for every page in every segment