Introduction to Paging

COMS W4118

References: Operating Systems Concepts (9e), Linux Kernel Development, previous W4118s
Copyright notice: care has been taken to use only those web images deemed by the instructor to be in the public domain. If you see a copyrighted image on any slide and are the copyright owner, please contact the instructor. It will be removed.
Multiple address spaces co-exist.

Logical view:
- AS1
- AS2
- AS3

Physical view:
- Physical view

0
Memory management wish-list

- **Sharing**
  - Multiple processes *coexist* in main memory

- **Transparency**
  - Processes *are not aware* that memory is shared

- **Protection**
  - Processes *cannot access* data of other processes or kernel

- **Efficiency**
  - Reasonable performance
  - *Do not waste* too much memory by fragmentation
Memory Management Unit (MMU)

- Translate program-generated logical address (virtual address) to real RAM address (physical address) at every reference
- Also check range and permissions
- Programmed by OS, executed by hardware
1. Segmentation unit
   – Segment & offset $\rightarrow$ linear address

2. Paging unit
   – Linear address $\rightarrow$ physical address

• Segmentation
  – Holdover from the olden days (16-bit x86 CPUs)
  – Used only minimally at this point
• Virtual address space divided into logical segments, each mapped to physical address region
x86 segmentation hardware

Logical address

selector

offset

Global descriptor table

base limit perm

Compute: base + offset
Check: offset <= limit
Check: permissions

Linear address

Compute: base + offset
Check: offset <= limit
Check: permissions
Paging overview

• **Goal**
  – Eliminate fragmentation due to large segments
  – Don’t allocate memory that will not be used
  – Enable fine-grained sharing

• **Paging:** divide memory into fixed-sized *pages*
  – For both virtual and physical memory

• **Another terminology**
  – A virtual page: *page*
  – A physical page: *frame*
• Address bits = page number + page offset
• Translate virtual page number (vpn) to physical page (frame) number (ppn/pfn) using page table

\[ \text{pa} = \text{page\_table}[\text{va/pg\_sz}] + \text{va}\%\text{pg\_sz} \]
Page translation example

Virtual Memory

Page 0
Page 1
Page 2
Page 3

Page table

0 1
1 4
2 3
3 7

Physical Memory

Page 0
Page 2
Page 1
Page 3
Page translation exercise

- 8-bit virtual address, 10-bit physical address, each page is 64 bytes

1. How many virtual pages?
   - $2^8 / 64 = 4$ virtual pages

2. How many physical pages?
   - $2^{10}/64 = 16$ physical pages

3. How many entries in page table?
   - Page table contains 4 entries

4. Given page table = [2, 5, 1, 8], what’s the physical address for virtual address 241?
   - 241 = 11110001b
   - 241/64 = 3 = 11b
   - 241%64 = 49 = 110001b
   - page_table[3] = 8 = 1000b
   - Physical address = 8 * 64 + 49 = 561 = 1000110001b
m-bit virtual address, n-bit physical address, k-bit page size

- # of virtual pages: \(2^{(m-k)}\)
- # of physical pages: \(2^{(n-k)}\)
- # of entries in page table: \(2^{(m-k)}\)

\[\text{vpn} = \text{va} / 2^k\]
\[\text{offset} = \text{va} \% 2^k\]
\[\text{ppn} = \text{page_table}[\text{vpn}]\]
\[\text{pa} = \text{ppn} \times 2^k + \text{offset}\]
Page protection

• Implemented by associating protection bits with each virtual page in page table

• Why do we need protection bits?

• Protection bits
  – present bit: map to a valid physical page?
  – read/write/execute bits: can read/write/execute?
  – user bit: can access in user mode?
  – x86: PTE_P, PTE_W, PTE_U

• Checked by MMU on each memory access
Page protection example

• What kind of pages?

Virtual Memory

Page 0
Page 1
Page 3

Page table

Physical Memory

0
1
2
3

1
4
3
7

101
110
000
111

pwu

Page 0
Page 1
Page 3
Implementation of page table

• Page table is stored in memory
  – Page table base register (PTBR) points to the base of page table
    • x86: cr3
  – OS stores base in process control block (PCB)
  – OS switches PTBR on each context switch

• Problem: each data/instruction access requires two memory accesses
  – Extra memory access for page table
Page table size issues

• Given:
  – A 32 bit address space (4 GB)
  – 4 KB pages
  – A page table entry of 4 bytes

• Implication: **page table is 4 MB per process!**

• Observation: **address space are often sparse**
  – Few programs use all of $2^{32}$ bytes

• Change page table structures to save memory
  – Hierarchical page tables