UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 537
Introduction to Operating Systems

Andrea C. Arpaci-Dusseau
Remzi H. Arpaci-Dusseau

Memory Management Continued

Questions answered in this lecture:

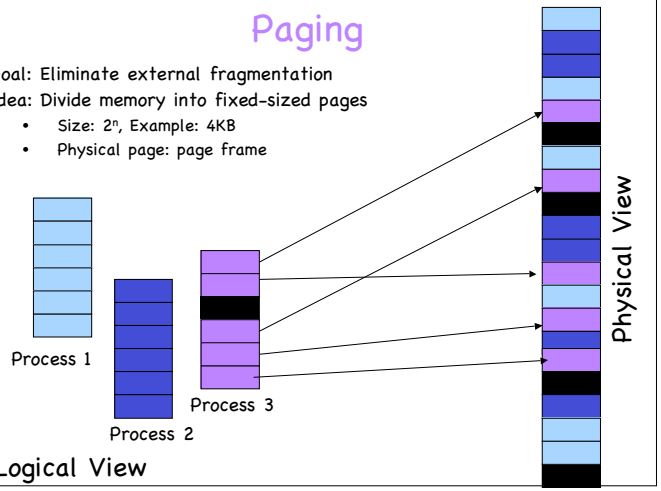
- What is paging?
- How can segmentation and paging be combined?
- How can one speed up address translation?

Paging

Goal: Eliminate external fragmentation

Idea: Divide memory into fixed-sized pages

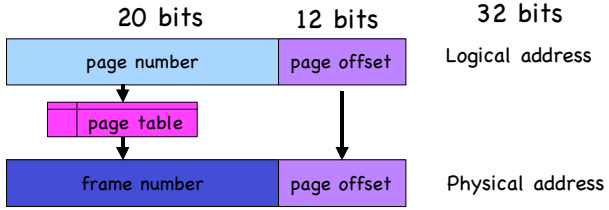
- Size: 2^n, Example: 4KB
- Physical page: page frame



Translation of Page Addresses

How to translate logical address to physical address?

- High-order bits of address designate page number
- Low-order bits of address designate offset within page



Page Table Implementation

Page table per process

- Page table entry (PTE) for each virtual page number (vpn)
  - frame number or physical page number (ppn)
  - R/W protection bits

Simple vpn->ppn mapping:

- No bounds checking, no addition
- Simply table lookup and bit substitution

How many entries in table?

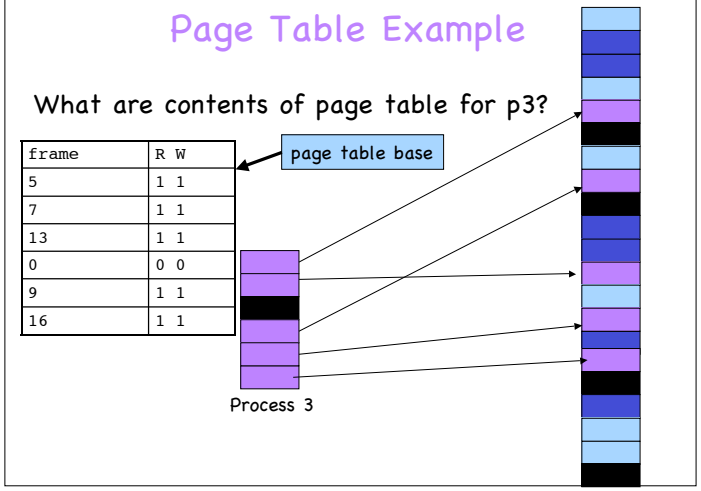
Can the page table reside in the MMU?

Track page table base in PCB, change on context-switch

Page Table Example

What are contents of page table for p3?

frame	R W
5	1 1
7	1 1
13	1 1
0	0 0
9	1 1
16	1 1



Advantages of Paging

No external fragmentation
- Any page can be placed in any frame in physical memory
- Fast to allocate and free
  - Alloc: No searching for suitable free space
  - Free: Doesn't have to coallesce with adjacent free space
  - Just use bitmap to show free/allocated page frames

Simple to swap-out portions of memory to disk
- Page size matches disk block size
- Can run process when some pages are on disk
- Add "present" bit to PTE

Enables sharing of portions of address space
- To share a page, have PTE point to same frame

Disadvantages of Paging

Internal fragmentation: Page size may not match size needed by process
- Wasted memory grows with larger pages
- Tension?

Additional memory reference to look up in page table --> Very inefficient
- Page table must be stored in memory
- MMU stores only base address of page table

Storage for page tables may be substantial
- Simple page table: Requires PTE for all pages in address space
  - Entry needed even if page not allocated
- Problematic with dynamic stack and heap within address space

Combine Paging and Segmentation

Goal: More efficient support for sparse address spaces

Idea:
- Divide address space into segments (code, heap, stack)
  - Segments can be variable length
- Divide each segment into fixed-sized pages

Logical address divided into three portions: System 370

Implementation
- Each segment has a page table
- Each segment track base (physical address) and bounds of page table (number of PTEs)
- What changes on a context-switch??

Example of Paging and Segmentation

Translate 24-bit logical to physical addresses

seg	base	bounds	R W
0	0x002000	0x14	1 0
1	0x000000	0x00	0 0
2	0x001000	0x0d	1 1

```
0x002070 read:
0x202016 read:
0x104c84 read:
0x010424 write:
0x210014 write:
0x203568 read:
```

...
0x01f
0x011
0x003
0x02a
0x013
...
0x00c
0x007
0x004
0x00b
0x006
...

0x002000

0x001000

Advantages of Paging and Segmentation

Advantages of Segments

- Supports sparse address spaces
  - Decreases size of page tables
  - If segment not used, not need for page table

Advantages of Pages

- No external fragmentation
- Segments can grow without any reshuffling
- Can run process when some pages are swapped to disk

Advantages of Both

- Increases flexibility of sharing
  - Share either single page or entire segment
  - How?

Disadvantages of Paging and Segmentation

Overhead of accessing memory

- Page tables reside in main memory
- Overhead reference for every real memory reference

Large page tables

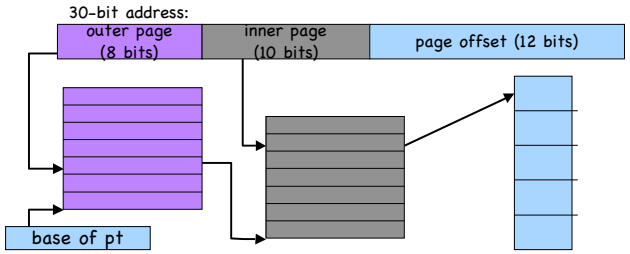
- Must allocate page tables contiguously
- More problematic with more address bits
- Page table size?
  - Assume 2 bits for segment, 18 bits for page number, 12 bits for offset

Page the Page Tables

Goal: Allow page tables to be allocated non-contiguously

Idea: Page the page tables

- Creates multiple levels of page tables
- Only allocate page tables for pages in use



Page the Page Tables Continued

How should logical address be structured?

- How many bits for each paging level?

Calculate such that page table fits within a page

- Goal: PTE size * number PTE = page size
- Assume PTE size = 4 bytes; page size = 4KB

  2^2 * number PTE = 2^12

  --> number PTE = 2^10

  →# bits for selecting inner page = 10

Apply recursively throughout logical address

Translation Look-Aside Buffer (TLB)

Goal: Avoid page table lookups in main memory

Idea: Hardware cache of recent page translations

- Typical size: 64 - 2K entries
- Index by segment + vpn --> ppn

Why does this work?

- process references few unique pages in time interval
- spatial, temporal locality

On each memory reference, check TLB for translation

- If present (hit): use ppn and append page offset
- Else (miss): Use segment and page tables to get ppn
  - Update TLB for next access (replace some entry)

How does page size impact TLB performance?