Virtual Memory 2

Kevin Walsh CS 3410, Spring 2010 Computer Science Cornell University

P & H Chapter 5.4-5

Virtual Memory Summary

PageTable for each process:

- 4MB contiguous in physical memory, or multi-level, ...
- every load/store translated to physical addresses

page table miss = page fault
 load the swapped-out page and retry instruction,
 or kill program if the page really doesn't exist,
 or tell the program it made a mistake

x86 Example: 2 level page tables, assume...

- 32 bit vaddr, 32 bit paddr
- 4k PDir, 4k PTables, 4k Pages
- Q:How many bits for a page number? A: 20
- Q: What is stored in each PageTableEntry?
- A: ppn, valid/dirty/r/w/x/...
- Q: What is stored in each PageDirEntry?
- A: ppn, valid/?/...
- Q: How many entries in a PageDirectory?
- A: 1024 four-byte PDEs
- Q: How many entires in each PageTable?
- A: 1024 four-byte PTEs



PTBR



x86 Example: 2 level page tables, assume...

- 32 bit vaddr, 32 bit paddr 4k PDir, 4k PTables, 4k Pages PTBR = 0x10005000 (physical)
- Write to virtual address 0x7192a44c... Q: Byte offset in page? PT Index?
- (1) PageDir is at 0x10005000, so...Fetch PDE from physical address 0x1005000+4*PDI
 - suppose we get {0x12345, v=1, ...}

(2) PageTable is at 0x12345000, so...Fetch PTE from physical address 0x12345000+4*PTI

- suppose we get {0x14817, v=1, d=0, r=1, w=1, x=0, ...}
- (3) Page is at 0x14817000, so...

Write data to physical address 0x1481744c Also: update PTE with d=1



PTBR



PD Index?

PTE

PTE

PTE

PTE

Virtual Memory Summary

PageTable for each process:

- 4MB contiguous in physical memory, or multi-level, ...
- every load/store translated to physical addresses
- page table miss: load a swapped-out page and retry instruction, or kill program

Performance?

- terrible: memory is already slow translation makes it slower
- Solution?
 - A cache, of course

Making Virtual Memory Fast The Translation Lookaside Buffer (TLB)

Hardware Translation Lookaside Buffer (TLB)

A small, very fast cache of recent address mappings

- TLB hit: avoids PageTable lookup
- TLB miss: do PageTable lookup, cache result for later





(1) Check TLB for vaddr (~ 1 cycle)

(2) TLB Hit

compute paddr, send to cache

(2) TLB Miss: traverse PageTables for vaddr

(3a) PageTable has valid entry for in-memory page

Load PageTable entry into TLB; try again (tens of cycles)

(3b) PageTable has entry for swapped-out (on-disk) page

• Page Fault: load from disk, fix PageTable, try again (millions of cycles)

(3c) PageTable has invalid entry

• Page Fault: kill process

TLB Coherency: What can go wrong?

- A: PageTable or PageDir contents change
 - swapping/paging activity, new shared pages, ...
- A: Page Table Base Register changes
 - context switch between processes











When PTE changes, PDE changes, PTBR changes.... Full Transparency: TLB coherency in hardware

- Flush TLB whenever PTBR register changes [easy – why?]
- Invalidate entries whenever PTE or PDE changes [hard – why?]
- TLB coherency in software
- If TLB has a no-write policy...
 - OS invalidates entry after OS modifies page tables
 - OS flushes TLB whenever OS does context switch

TLB parameters (typical)

- very small (64 256 entries), so very fast
- fully associative, or at least set associative
- tiny block size: why?

Intel Nehalem TLB (example)

- 128-entry L1 Instruction TLB, 4-way LRU
- 64-entry L1 Data TLB, 4-way LRU
- 512-entry L2 Unified TLB, 4-way LRU

Virtual Memory meets Caching Virtually vs. physically addressed caches Virtually vs. physically tagged caches Q: Can we remove the TLB from the critical path? A: Virtually-Addressed Caches





Q: What happens on context switch?Q: What about virtual memory aliasing?Q: So what's wrong with physically addressed caches?

Physically-Addressed Cache

- slow: requires TLB (and maybe PageTable) lookup first
 Virtually-Addressed Cache
 - fast: start TLB lookup before cache lookup finishes
 - PageTable changes (paging, context switch, etc.)

 \rightarrow need to purge stale cache lines (how?)

Synonyms (two virtual mappings for one physical page)
 → could end up in cache twice (very bad!)

Virtually-Indexed, Physically Tagged Cache

- ~fast: TLB lookup in parallel with cache lookup
- PageTable changes \rightarrow no problem: phys. tag mismatch
- Synonyms \rightarrow search and evict lines with same phys. tag



Typical L1: On-chip virtually addressed, physically tagged Typical L2: On-chip physically addressed Typical L3: On-chip ...

Caches, Virtual Memory, & TLBs

Where can block be placed?

- Direct, n-way, fully associative
- What block is replaced on miss?
 - LRU, Random, LFU, ...
- How are writes handled?
 - No-write (w/ or w/o automatic invalidation)
 - Write-back (fast, block at time)
 - Write-through (simple, reason about consistency)

	L1	Paged Memory	TLB
Size (blocks)	1/4k to 4k	16k to 1M	64 to 4k
Size (kB)	16 to 64	1M to 4G	2 to 16
Block size (B)	16-64	4k to 64k	4-32
Miss rates	2%-5%	10 ⁻⁴ to 10 ⁻⁵ %	0.01% to 2%
Miss penalty	10-25	10M-100M	100-1000