SE350: Operating Systems

Lecture 8: Efficient Address Translation
Main Points

• Efficient Address Translation
  • Translation lookaside buffers (TLB)
  • Virtually and physically addressed caches
Address Translation Concept

- Translator converts (virtual) addresses generated by programs into physical memory addresses
- Different processor architectures store page tables differently in hardware
- Kernel keeps its own data structure for address translation
Efficient Address Translation

• How can we reduce overhead of address translation?
  • Cache recent virtual to physical page translations

• Translation lookaside buffer (TLB)
  • If cache hit, use the corresponding TLB entry
  • If cache miss, walk multi-level page table

• Cost of translation
  TLB lookup cost + Pr(TLB miss) × Page table lookup cost
TLB and Page Table Translation

- TLB is implemented in fast, on-chip static memory, next to the processor.
- TLB lookup is much faster than doing a full address translation.
- Hardware cost of a TLB is modest compared to performance gain.
TLB Lookup

- Each virtual page number is checked against all TLB entries
- If there is a match, physical page frame and permissions are found
- If not, the hardware multi-level page table lookup is invoked
- Note the hardware page tables are omitted from the picture
When Do TLBs Work/Not Work?

• For HD displays, video frame buffer could be large
  • E.g., 4k display: 32 bits × 4K × 3K = 48MB (spans 12K of 4KB pages)
• Even large on-chip TLB with 256 entries cannot cover entire display
• Each horizontal line of pixels could be on a page
• Drawing a vertical line could require loading a new TLB entry
Superpages: Improve TLB Hit Rate

- Reduce number of TLB entries for large, contiguous regions of memory
  - Represent 2 adjacent 4KB pages by single 8KB superpage
- By setting a flag, TLB entry can be a page or a superpage
  - E.g., in x86: 4KB (12 bits offset), 2MB (21 bits offset), or 1GB (30 bits offset)
MIPS Software Loaded TLB

- Software defined translation tables
  - If TLB hit, use the corresponding TLB entry
  - If TLB miss, trap to kernel
  - Kernel fills TLB with translation and resumes execution

- Flexible kernel-level page translation
  - Page tables
  - Multi-level page tables
  - Inverted page tables
Questions

• What is the benefit of software loaded TLB?
  • Convenient for OS
    • No need to keep track of two sets of page tables, one of hardware and one for itself

• What is the downside of software loaded TLB?
  • Lower performance for applications
    • In modern x86 processor, TLB miss could be resolved in the equivalent of 17 instructions, whereas kernel trap could take hundreds or thousands of instructions to process
TLB Consistency

- Address translations are stored in multiple places
  - TLB
  - Hardware (multi-level) page tables
  - Kernel (portable) data structures
- OS must ensure consistency of these copies
- There are three main issues
  - What should happen on a process context switch?
  - What should happen when OS modifies page table entry?
  - What should happen in a multiprocessor where each processor has its own TLB?
Process Context Switch

• Old process virtual address translations are not valid
• Page table register points to new process’s page table

• What should happen to TLB entries?
  • **Flush** the TLB (discard its content)
  • Tag TLB entries with process ID (tagged TLB)
    • Hit if addresses match and process ID matches current process
Permission Reduction

• Keeping TLB consistent with page table is OS’s responsibility

• Nothing needs to be done when permission is added
  • E.g., changing invalid to read-only
  • Any reference would cause exception, OS re-loads TLB

• If permission is reduced, TLB should be updated
  • Early computers discarded the entire content of TLB
  • Modern architectures (e.g., x86 and ARM) support removal of individual entries
Suppose processor 1 wants to update entry for page 0x53 in process 0

- First, it must remove the entry from its TLB
- Then, it must send an interprocessor interrupt to each processor requesting it to remove the old translation

Shootdown is complete only when all processors verify that the old translation has been removed

TLB shootdown overhead increases linearly with the # of processors
Improve Efficiency Even More!

• TLB improves performance by caching recent translations

• How to improve performance even more?
  • Add another layer of cache!

• What is the cost of a first level TLB miss?
  • Second level TLB lookup

• What is the cost of a second level TLB miss?
  • x86: 2-4 level page table walk
Improve Efficiency Even More: Virtually Addressed Cache

- Too slow to first access TLB to find physical address, then look up address in the memory
- Instead, add a virtually addressed cached
- In parallel, access TLB to generate physical address in case of a cache miss
Aliasing

• Multiple virtual cache entries could refer to the same physical memory
• When one process modifies its copy; how does the system know to update the other copy?
• Typical solution
  • Keep both virtual and physical address for each entry in virtually addressed cache
  • Lookup virtually addressed cache and TLB in parallel
  • Check if physical address from TLB matches multiple entries, and update/invalidate other copies
• Add a physically addressed cache after the virtually addressed cache and TLB, and before main memory
Acknowledgment

• This lecture is a slightly modified version of the one prepared by Tom Anderson