Administrivia

Project3 available now

• Design Doc due next week, Monday, April 16\textsuperscript{th}
• Schedule a Design Doc review Mtg now for next week
• Whole project due Monday, April 23\textsuperscript{rd}
• Competition/Games night Friday, April 27\textsuperscript{th}, 5-7pm

Prelim3 is in two and a half weeks, Thursday, April 26\textsuperscript{th}

• Time and Location: 7:30pm in Olin Hall room 155
• Old prelims are online in CMS
Goals for Today

Virtual Memory

- Address Translation
  - Pages, page tables, and memory mgmt unit
- Paging
- Role of Operating System
  - Context switches, working set, shared memory
- Performance
  - How slow is it
  - Making virtual memory fast
  - Translation lookaside buffer (TLB)
- Virtual Memory Meets Caching
Virtual Memory
Big Picture: Multiple Processes

How to Run multiple processes?

*Time-multiplex* a single CPU core (multi-tasking)
- Web browser, skype, office, ... all must co-exist

Many cores per processor (multi-core)
- or many processors (multi-processor)
- Multiple programs run *simultaneously*
Big Picture: (Virtual) Memory

Memory: big & slow vs Caches: small & fast
Big Picture: (Virtual) Memory

Memory: big & slow vs Caches: small & fast

Processor

\[
\begin{align*}
& \text{LB } \$1 \leftarrow M[1] \\
& \text{LB } \$2 \leftarrow M[5] \\
& \text{LB } \$3 \leftarrow M[1] \\
& \text{LB } \$3 \leftarrow M[4] \\
& \text{LB } \$2 \leftarrow M[0] \\
& \text{LB } \$2 \leftarrow M[12] \\
& \text{LB } \$2 \leftarrow M[5] \\
& \text{LB } \$2 \leftarrow M[12] \\
& \text{LB } \$2 \leftarrow M[5] \\
& \text{LB } \$2 \leftarrow M[5] \\
& \text{LB } \$2 \leftarrow M[5] \\
\end{align*}
\]

Cache

Memory

0xffff...f
0x7ff...f
Stack

Heap

Data

Text

0x000...0

Misses:

Hits:
Processor & Memory

CPU address/data bus...
... routed through caches
... to main memory
• Simple, fast, but...

Q: What happens for LW/SW to an invalid location?
• 0x000000000 (NULL)
• uninitialized pointer

A: Need a memory management unit (MMU)
• Throw (and/or handle) an exception
Multiple Processes

Q: What happens when another program is executed concurrently on another processor?

A: The addresses will conflict
• Even though, CPUs may take turns using memory bus
Multiple Processes

Q: Can we relocate second program?
Solution? Multiple processes/processors

Q: Can we relocate second program?
A: Yes, but...

- What if they don’t fit?
- What if not contiguous?
- Need to recompile/relink?
- ...

![Diagram of CPU, Memory, Stack, Data, Heap layers]
All problems in computer science can be solved by another level of indirection.

\[
paddr = \text{PageTable}[vaddr];
\]

– David Wheeler  
– or, Butler Lampson  
– or, Leslie Lamport  
– or, Steve Bellovin
Performance
Performance Review

Virtual Memory Summary

Page Table 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
Beyond Flat Page Tables
Assume most of PageTable is empty

How to translate addresses? Multi-level PageTable

* x86 does exactly this
Page Table Review

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 physical 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 entries in each PageTable?
A: 1024 four-byte PTEs
Page Table Review Example

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? PD 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
Performance Review

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)
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
TLB Diagram

V  R  W  X  D  tag  ppn

V  R  W  X  D
0  0  0  0  0  invalid
1  0  0  0  0  invalid
1  0  0  0  0  invalid
0  1  0  0  0  invalid
0  0  0  0  0  invalid
0  0  0  0  0  invalid
0  1  1  1  1  invalid
0  1  1  1  1  invalid
0  1  1  1  1  invalid
0  1  1  1  1  invalid
A TLB in the Memory Hierarchy

(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

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
Translation Lookaside Buffers (TLBs)

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

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
Virtually Addressed Caching

Q: Can we remove the TLB from the critical path?

A: Virtually-Addressed Caches

Diagram:
- CPU
- TLB Lookup
- Virtually Addressed Cache
- PageTable Lookup
- Mem
- Disk
Virtual vs. Physical Caches

Cache works on physical addresses

Cache works on virtual addresses

Q: What happens on context switch?
Q: What about virtual memory aliasing?
Q: So what’s wrong with physically addressed caches?
Indexing vs. Tagging

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.)
  → 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 → no problem: phys. tag mismatch
• Synonyms → search and evict lines with same phys. tag
Indexing vs. Tagging
Typical Cache Setup

Typical L1: On-chip virtually addressed, physically tagged
Typical L2: On-chip physically addressed
Typical L3: On-chip ...
Summary of Caches/TLBs/VM

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)
Summary of Caches/TLBs/VM

Caches, Virtual Memory, & TLBs

Where can block be placed?
- Caches: direct/n-way/fully associative (fa)
- VM: fa, but with a table of contents to eliminate searches
- TLB: fa

What block is replaced on miss?
- varied

How are writes handled?
- Caches: usually write-back, or maybe write-through, or maybe no-write w/ invalidation
- VM: write-back
- TLB: usually no-write
## Summary of Cache Design Parameters

<table>
<thead>
<tr>
<th></th>
<th>L1</th>
<th>Paged Memory</th>
<th>TLB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (blocks)</td>
<td>1/4k to 4k</td>
<td>16k to 1M</td>
<td>64 to 4k</td>
</tr>
<tr>
<td>Size (kB)</td>
<td>16 to 64</td>
<td>1M to 4G</td>
<td>2 to 16</td>
</tr>
<tr>
<td>Block size (B)</td>
<td>16-64</td>
<td>4k to 64k</td>
<td>4-32</td>
</tr>
<tr>
<td>Miss rates</td>
<td>2%-5%</td>
<td>$10^{-4}$ to $10^{-5}$%</td>
<td>0.01% to 2%</td>
</tr>
<tr>
<td>Miss penalty</td>
<td>10-25</td>
<td>10M-100M</td>
<td>100-1000</td>
</tr>
</tbody>
</table>