Review: Caches

- Cache design choices:
  - size of cache: speed v. capacity
  - direct-mapped v. associative
  - for N-way set assoc: choice of N
  - block replacement policy
  - 2nd level cache?
  - Write through v. write back?
- Best choice depends on programs, technology, budget.
- Use performance model to pick between choices.

Another View of the Memory Hierarchy

Memory Hierarchy Requirements

- If Principle of Locality allows caches to offer (close to) speed of cache memory with size of DRAM memory, then recursively why not use at next level to give speed of DRAM memory, size of Disk memory?
- While we’re at it, what other things do we need from our memory system?

Virtual Memory

- Called “Virtual Memory”
- Next level in the memory hierarchy:
  - Provides program with illusion of a very large main memory:
  - Working set of “pages” reside in main memory - others reside on disk.
- Also allows OS to share memory, protect programs from each other
- Today, more important for protection vs. just another level of memory hierarchy
- Each process thinks it has all the memory to itself

(Historically, it predates caches)
Virtual to Physical Address Translation

- Each program operates in its own virtual address space; only program running
- Each is protected from the other
- OS can decide where each goes in memory
- Hardware (HW) provides virtual $\rightarrow$ physical mapping

**Analogy**

- Book title like virtual address
- Library of Congress call number like physical address
- Card catalogue like page table, mapping from book title to call #
- On card for book, in local library vs. in another branch like valid bit indicating in main memory vs. on disk
- On card, available for 2-hour in library use (vs. 2-week checkout) like access rights

**Simple Example: Base and Bound Reg**

- $\textit{base}$ and $\textit{bound}$
- User C
- Enough space for User D, but discontinuous (“fragmentation problem”)
- Want:
  - discontinuous mapping
  - Process size $\gg$ mem
- Addition not enough! $\Rightarrow$ use Indirection!

**Mapping Virtual Memory to Physical Memory**

- Divide into equal sized chunks (about 4 KB - 8 KB)
- Any chunk of Virtual Memory assigned to any chunk of Physical Memory (“page”)

**Paging Organization (assume 1 KB pages)**

- Cannot have simple function to predict arbitrary mapping
- Use table lookup of mappings
- Use table lookup (“Page Table”) for mappings: Page number is index
- Virtual Memory Mapping Function
  - Physical Offset = Virtual Offset
  - Physical Page Number = PageTable[Virtual Page Number]
  - (P.P.N. also called “Page Frame”)

**Virtual Memory Mapping Function**

<table>
<thead>
<tr>
<th>Physical Address</th>
<th>Page is unit of mapping</th>
<th>Virtual Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>page 0 1K</td>
<td>page 0 1K</td>
</tr>
<tr>
<td>1024</td>
<td>page 1 1K</td>
<td>page 1 1K</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>7168</td>
<td>page 7 1K</td>
<td>page 31 1K</td>
</tr>
<tr>
<td></td>
<td>1K</td>
<td>1K</td>
</tr>
</tbody>
</table>

**Virtual Memory**

- $\textit{addr}$
- $\textit{trans}$
- $\textit{map}$
- $\textit{phys}$
- $\textit{inst}$
- $\textit{load}$
- $\textit{store}$

- $\textit{OS}$
- $\textit{User A}$
- $\textit{User B}$
- $\textit{User C}$
- $\textit{User D}$

- $\textit{heap}$
- $\textit{static}$
- $\textit{code}$
- $\textit{heap}$
- $\textit{static}$

- $\textit{mem}$
- $\textit{inst}$
- $\textit{load}$
- $\textit{store}$

- $\textit{main}$
- $\textit{disk}$
- $\textit{in}$
- $\textit{on}$

- $\textit{valid}$
- $\textit{access}$
Address Mapping: Page Table

Virtual Address:
  page no. offset
  Page Table
  Base Reg
  index into page table

Page Table located in physical memory

Page Table

- A page table is an operating system structure which contains the mapping of virtual addresses to physical locations
  - There are several different ways, all up to the operating system, to keep this data around
  - Each process running in the operating system has its own page table
    - “State” of process is PC, all registers, plus page table
    - OS changes page tables by changing contents of Page Table Base Register

Administivia

- Do your reading! Caches, VM can be tricky to get.
- We’ve hired two more readers to help; here is the status of your assignments

Requirements revisited

Remember the motivation for VM:
- Sharing memory with protection
  - Different physical pages can be allocated to different processes (sharing)
  - A process can only touch pages in its own page table (protection)
- Separate address spaces
  - Since programs work only with virtual addresses, different programs can have different data/code at the same address!

What about the memory hierarchy?

Page Table Entry (PTE) Format

- Contains either Physical Page Number or indication not in Main Memory
  - OS maps to disk if Not Valid (V = 0)

- If valid, also check if have permission to use page; Access Rights (A.R.) may be Read Only, Read/Write, Executable

Paging/Virtual Memory Multiple Processes

User A:
  Virtual Memory
  Physical Memory
  Stack
  Code

User B:
  Virtual Memory
  Physical Memory
  Stack
  Code

64 MB

Page Table

A Page Table

B Page Table
Comparing the 2 levels of hierarchy

<table>
<thead>
<tr>
<th>Cache version</th>
<th>Virtual Memory vers.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block or Line</td>
<td>Page</td>
</tr>
<tr>
<td>Miss</td>
<td>Page Fault</td>
</tr>
<tr>
<td>Block Size: 32-64B</td>
<td>Page Size: 4K-8KB</td>
</tr>
<tr>
<td>Placement:</td>
<td>Fully Associative</td>
</tr>
<tr>
<td>Replacement:</td>
<td>Least Recently Used</td>
</tr>
<tr>
<td>Write Thru or Back</td>
<td>Write Back</td>
</tr>
</tbody>
</table>

Notes on Page Table

- Solves Fragmentation problem: all chunks same size, so all holes can be used
- OS must reserve “Swap Space” on disk for each process
- To grow a process, ask Operating System
  - If unused pages, OS uses them first
  - If not, OS swaps some old pages to disk
    - (Least Recently Used to pick pages to swap)
- Each process has own Page Table
- Will add details, but Page Table is essence of Virtual Memory

Virtual Memory Problem #1

- Map every address ⇒ 1 indirection via Page Table in memory per virtual address ⇒ 1 virtual memory accesses = 2 physical memory accesses ⇒ SLOW!
- Observation: since locality in pages of data, there must be locality in virtual address translations of those pages
- Since small is fast, why not use a small cache of virtual to physical address translations to make translation fast?
- For historical reasons, cache is called a Translation Lookaside Buffer, or TLB

Translation Look-Aside Buffers (TLBs)

- TLBs usually small, typically 128 - 256 entries
- Like any other cache, the TLB can be direct mapped, set associative, or fully associative

On TLB miss, get page table entry from main memory

Peer Instruction

A. Locality is important yet different for cache and virtual memory (VM): temporal locality for caches but spatial locality for VM
B. Cache management is done by hardware (HW), page table management by the operating system (OS), but TLB management is either by HW or OS
C. VM helps both with security and cost

And in conclusion...

- Manage memory to disk? Treat as cache
  - Included protection as bonus, now critical
  - Use Page Table of mappings for each user vs. tag/data in cache
- TLB is cache of Virtual → Physical addr trans
- Virtual Memory allows protected sharing of memory between processes
- Spatial Locality means Working Set of Pages is all that must be in memory for process to run fairly well