Netsky/Bagle/MyDoom War ⇒ "Hey, Netsky... don't ruine (sp) our bussiness (sp), wanna start a war?" [Bagle.J worm's code]
Review…

- 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 Page Table

- 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
Locality…

• Spatial Locality != Temporal Locality
  • SL: If I access X, I’ll (probably) stay close
  • TL: If I access X, I’ll (probably) get back soon

• Normally both hold, but…
  • Sequential memory access: SL, not TL
  • Lab8 experiment, large stride: TL, not SL
Peer Instructions

A. Locality is important yet different for cache and virtual memory (VM): temporal locality for caches but spatial locality for VM

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>6</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>7</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>8</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

C. VM helps both with security and cost
Virtual Memory Problem #1

• Slow:
  • Every memory access requires:
    - 1 access to PT to get VPN->PPN translation
    - 1 access to MEM to get data at PA

• Solution:
  • Cache the Page Table
    - Make common case fast
    - PT cache called “TLB”
  • “block size” is just 1 VA->PA mapping
  • TLB associativity?
Virtual Memory Problem #2

• **Page Table too big!**
  - 4GB Virtual Memory ÷ 1 KB page
    ⇒ ~ 4 million Page Table Entries
    ⇒ 16 MB just for Page Table for 1 process,
    8 processes ⇒ 256 MB for Page Tables!

• Spatial Locality to the rescue
  - Each page is 4 KB, lots of nearby references
  - But large page size wastes resources

• No matter how big program is, at any time only accessing a few pages
  - “**Working Set**”: recently used pages
Solutions

- Page the Page Table itself!
  - Works, but must be careful with never-ending page faults
  - Pin some PT pages to memory

- 2-level page table

- Solutions tradeoff in-memory PT size for slower TLB miss
  - Make TLB large enough, highly associative so rarely miss on address translation
  - CS 162 will go over more options and in greater depth
### Page Table Shrink:

- **Single Page Table**
  
<table>
<thead>
<tr>
<th>Page Number</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 bits</td>
<td>12 bits</td>
</tr>
</tbody>
</table>

- **Multilevel Page Table**
  
<table>
<thead>
<tr>
<th>Super Page No.</th>
<th>Page Number</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 bits</td>
<td>10 bits</td>
<td>12 bits</td>
</tr>
</tbody>
</table>

- Only have second level page table for valid entries of super level page table

- Exercise 7.35 explores exact space savings
Three Advantages of Virtual Memory

1) Translation:

- Program can be given consistent view of memory, even though physical memory is scrambled (illusion of contiguous memory)
- All programs starting at same set address
- Illusion of ~ infinite memory ($2^{32}$ or $2^{64}$ bytes)
- Makes multiple processes reasonable

- Only the most important part of program (“Working Set”) must be in physical memory
- Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later
Three Advantages of Virtual Memory

- **User program view of memory:**
  - Contiguous
  - Start from some set address
  - Infinitely large
  - Is the only running program

- **Reality:**
  - Non-contiguous
  - Start wherever available memory is
  - Finite size
  - Many programs running at a time
Three Advantages of Virtual Memory

2) Protection:
   • Different processes protected from each other
   • Different pages can be given special behavior
     - (Read Only, Invisible to user programs, etc).
   • Kernel data protected from User programs
   • Very important for protection from malicious programs
   • Special Mode in processor ("Kernel more") allows processor to change page table/TLB

3) Sharing:
   • Can map same physical page to multiple users ("Shared memory")
Virtual Memory Overview

• Let’s say we’re fetching some data:
  • Check TLB (input: VPN, output: PPN)
    - hit: fetch translation
    - miss: check page table (in memory)
      – page table hit: fetch translation
      – page table miss: page fault, fetch page from disk to memory, return translation to TLB
  • Check cache (input: PPN, output: data)
    - hit: return value
    - miss: fetch value from memory
Peer Instruction

1. Increasing at least one of \{associativity, block size\} is always a win

2. Higher DRAM bandwidth translates to a lower miss rate

3. DRAM access time improves roughly as fast as density

A: 1 2 TRUE
B: 1 3 TRUE
C: 2 3 TRUE
D: 1 TRUE
E: 2 TRUE
F: ALL FALSE
Address Translation & 3 Concept tests

Virtual Address

VPN | INDEX | Offset

TLB

VPN | INDEX | Offset

Data Cache

Tag | Data

Physical Address

TAG | INDEX | Offset
Peer Instruction (1/3)

• 40-bit virtual address, 16 KB page

Virtual Page Number (\(?\) bits) \hspace{1cm} Page Offset (\(?\) bits)

• 36-bit physical address

Physical Page Number (\(?\) bits) \hspace{1cm} Page Offset (\(?\) bits)

• Number of bits in Virtual Page Number/ Page offset, Physical Page Number/Page offset?

A: \hspace{1cm} 22/18 (VPN/PO), 22/14 (PPN/PO)
B: \hspace{1cm} 24/16, 20/16
C: \hspace{1cm} 26/14, 22/14
D: \hspace{1cm} 26/14, 26/10
E: \hspace{1cm} 28/12, 24/12
Peer Instruction (2/3): 40b VA, 36b PA

- 2-way set-assoc. TLB, 256 entries, 40b VA:

  TLB Tag (? bits) | TLB Index (? bits) | Page Offset (14 bits)

- TLB Entry: Valid bit, Dirty bit, Access Control (say 2 bits), Virtual Page Number, Physical Page Number

  V  D  Access (2 bits)  TLB Tag (? bits)  Physical Page No. (? bits)

- Number of bits in TLB Tag / Index / Entry?

  A: 12 / 14 / 38 (TLB Tag / Index / Entry)
  B: 14 / 12 / 40
  C: 19 / 7 / 45
  D: 19 / 7 / 58
Peer Instruction (3/3)

- 2-way set-assoc, 64KB data cache, 64B block

<table>
<thead>
<tr>
<th>Cache Tag (? bits)</th>
<th>Cache Index (? bits)</th>
<th>Block Offset (? bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Page Address (36 bits)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Data Cache Entry: Valid bit, Dirty bit, Cache tag + ? bits of Data

| V | D | Cache Tag (? bits) | Cache Data (? bits) |

- Number of bits in Data cache Tag / Index / Offset / Entry?

| A: | A: | 12 / 9 / 14 / 87 (Tag/Index/Offset/Entry) |
| B: | B: | 20 / 10 / 6 / 86 |
| C: | C: | 20 / 10 / 6 / 534 |
| D: | D: | 21 / 9 / 6 / 87 |
| E: | E: | 21 / 9 / 6 / 535 |
$&VM Review: 4 Qs for any Mem. Hierarchy

• Q1: Where can a block be placed in the upper level? *(Block placement)*

• Q2: How is a block found if it is in the upper level? *(Block identification)*

• Q3: Which block should be replaced on a miss? *(Block replacement)*

• Q4: What happens on a write? *(Write strategy)*
Q1: Where block placed in upper level?

- Block 12 placed in 8 block cache:
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Mod Number Sets

Fully associative:
block 12 can go anywhere

Direct mapped:
block 12 can go only into block 4 (12 mod 8)

Set associative:
block 12 can go anywhere in set 0 (12 mod 4)
Q2: How is a block found in upper level?

- Direct indexing (using index and block offset), tag compares, or combination
- Increasing associativity shrinks index, expands tag
Q3: Which block replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

<table>
<thead>
<tr>
<th>Size</th>
<th>LRU</th>
<th>Ran</th>
<th>LRU</th>
<th>Ran</th>
<th>LRU</th>
<th>Ran</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 KB</td>
<td>5.2%</td>
<td>5.7%</td>
<td>4.7%</td>
<td>5.3%</td>
<td>4.4%</td>
<td>5.0%</td>
</tr>
<tr>
<td>64 KB</td>
<td>1.9%</td>
<td>2.0%</td>
<td>1.5%</td>
<td>1.7%</td>
<td>1.4%</td>
<td>1.5%</td>
</tr>
<tr>
<td>256 KB</td>
<td>1.15%</td>
<td>1.17%</td>
<td>1.13%</td>
<td>1.13%</td>
<td>1.12%</td>
<td>1.12%</td>
</tr>
</tbody>
</table>
Q4: What to do on a write hit?

- **Write-through**
  - update the word in cache block and corresponding word in memory

- **Write-back**
  - update word in cache block
  - allow memory word to be “stale”

  => add ‘dirty’ bit to each line indicating that memory be updated when block is replaced

  => OS flushes cache before I/O !!!

- **Performance trade-offs?**
  - **WT:** read misses cannot result in writes
  - **WB:** no writes of repeated writes
In Conclusion…

- Virtual memory to Physical Memory Translation too slow?
  - Add a cache of Virtual to Physical Address Translations, called a **TLB**
  - Need more compact representation to reduce memory size cost of simple 1-level page table (especially 32- ⇒ 64-bit address)

- Spatial Locality means Working Set of Pages is all that must be in memory for process to run fairly well

- Virtual Memory allows protected sharing of memory between processes with less swapping to disk