JAILBREAK YOUR IPHONE BY VISITING A WEBSITE
http://preview.tinyurl.com/2b2qfhy

A new jailbreak technique for all iOS devices (iPhone, iPad, iPod, etc) has been released that allows you to jailbreak (or root) your device by simply visiting a website. This seems awesome... until you realize that by visiting that a website, you end up rewriting portions of your operating system via an exploit in the pdf viewer. It's only a matter of time before malicious parties use the same techniques to do nefarious things. Be careful where you visit.

And in review...

• Virtual memory!
  - Provides each process with the illusion of a large main memory all to itself.
  - Included protection as bonus, now critical
• Use Page Table of mappings for each process
• TLB is cache of Virtual ⇒ Physical addr trans (cache of PTEs)
• All that must be in memory for a process to run fairly well is the Working Set of Pages. This is Spatial Locality.
• Virtual Memory allows protected sharing of memory between processes

Address Mapping: Page Table

Virtual Address: Page Table

Page Table

Valid Access Rights Physical Page Address

Page Table located in physical memory

Fetching data on a memory read

- 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, remember it in cache, return value

Typical TLB Format

<table>
<thead>
<tr>
<th>Tag</th>
<th>Physical Page #</th>
<th>Dirty</th>
<th>Ref</th>
<th>Valid</th>
<th>Access Rights</th>
</tr>
</thead>
</table>

- TLB just a cache on the page table mappings
- TLB access time comparable to cache (much less than main memory access time)
- Dirty: since use write back, need to know whether or not to write page to disk when replaced
- Ref: Used to help calculate LRU on replacement
  - Cleared by OS periodically, then checked to see if page was referenced
What if not in TLB?

- Option 1: Hardware checks page table and loads new Page Table Entry into TLB
- Option 2: Hardware traps to OS, up to OS to decide what to do
  - MIPS follows Option 2: Hardware knows nothing about page table, just knows about TLB format.
  - A trap is a synchronous exception in a user process, often resulting in the OS taking over and performing some action before returning to the program.
    - More about exceptions next lecture

What if the data is on disk?

- We load the page off the disk into a free block of memory, using a DMA transfer (Direct Memory Access – special hardware support to avoid processor)
  - Meantime we switch to some other process waiting to be run
  - When the DMA is complete, we get an interrupt and update the process’s page table
    - So when we switch back to the task, the desired data will be in memory

What if we don’t have enough memory?

- We chose some other page belonging to a program and transfer it onto the disk if it is dirty
  - If clean (disk copy is up-to-date), just overwrite that data in memory
  - We chose the page to evict based on replacement policy (e.g., LRU)
  - And update that program’s page table to reflect the fact that its memory moved somewhere else
  - If continuously swap between disk and memory, called Thrashing

We’re done with new material

Let’s now review w/Questions

4 Qs for any Memory Hierarchy

- Q1: Where can a block be placed?
  - One place (direct mapped)
  - A few places (set associative)
  - Any place (fully associative)
- Q2: How is a block found?
  - Indexing (as in a direct-mapped cache)
  - Limited search (as in a set-associative cache)
  - Full search (as in a fully associative cache)
  - Separate lookup table (as in a page table)
- Q3: Which block is replaced on a miss?
  - Least recently used (LRU)
  - Least frequently used (LFU)
  - Random
- Q4: How are writes handled?
  - Write through (Level never inconsistent w/lower)
  - Write back (Could be “dirty”, must have dirty bit)

Q1: Where block placed in upper level?

- Block #12 placed in 8 block cache:
  - Fully associative
  - Direct mapped
  - 2-way set associative
    - Set Associative Mapping = Block # Mod # of Sets
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>
</tr>
</thead>
<tbody>
<tr>
<td>16 KB</td>
<td>5.2%</td>
<td>5.7%</td>
<td>4.7%</td>
<td>5.3%</td>
</tr>
<tr>
<td>64 KB</td>
<td>1.9%</td>
<td>2.0%</td>
<td>1.5%</td>
<td>1.7%</td>
</tr>
<tr>
<td>256 KB</td>
<td>1.15%</td>
<td>1.17%</td>
<td>1.13%</td>
<td>1.13%</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

Administrivia

- Reminder: Sign up for a grading slot for Project 2 if you haven’t already!
- Project 3 is due Monday, August 9th at midnight.
- Final exam is Thursday, August 12th from 8am-11am in 10 Evans
- Please register your iclicker so you can receive credit for your votes!
  - http://www.iclicker.com/registration/
  - If you’ve registered before, do so again!
  - You must enter your correct student ID number and name, or else I won’t get your data!

Three Advantages of Virtual Memory

1) Translation:
   - Program can be given consistent view of memory, even though physical memory is scrambled
   - 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

2) Protection:
   - Different processes protected from each other
   - Different pages can be given special behavior
     - (Read Only, Invisible to user programs, etc).
   - OS (we sometimes call the OS code the "Kernel") data protected from User programs

3) Sharing:
   - Can map same physical page to multiple users ("Shared memory")
Virtual Memory Overview (1/2)

- User program view of memory:
  - Contiguous
  - Start from some set address
  - Infinitely large (almost...)
  - Is the only running program

- Reality:
  - Non-contiguous
  - Start wherever available memory is
  - Finite size
  - Many programs running at a time

Question (1/3)

- 40-bit virtual address, 16 KB page

<table>
<thead>
<tr>
<th>Virtual Page Number (?? bits)</th>
<th>Page Offset (?? bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Page Number (?? bits)</td>
<td>Page Offset (?? bits)</td>
</tr>
</tbody>
</table>

- 36-bit physical address

Question (2/3): 40b VA, 36b PA

- 2-way set-associ. TLB, 512 entries, 40b VA:

<table>
<thead>
<tr>
<th>TLB Tag (?? bits)</th>
<th>TLB Index (?? bits)</th>
<th>Page Offset (14 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual Page Number (26 bits)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>V</th>
<th>D</th>
<th>Access (2 bits)</th>
<th>TLB Tag (?? bits)</th>
<th>Physical Page No. (?? bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: 24/16, 20/16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B: 26/14, 22/14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C: 26/14, 26/10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D: 26/12, 24/12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Question (3/3)

- 2-way set-associ 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 + ?? Bytes of Data

<table>
<thead>
<tr>
<th>V</th>
<th>D</th>
<th>Cache Tag (?? bits)</th>
<th>Cache Data (?? bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: 12 / 9 / 14 / 87 (Tag/Index/Offset/Entry)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2: 20 / 10 / 6 / 86</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3: 20 / 10 / 6 / 534</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4: 21 / 9 / 6 / 87</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5: 21 / 9 / 6 / 535</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

And in Conclusion...

- Virtual memory to Physical Memory Translation too slow?
  - Add a cache of Virtual to Physical Address Translations, called a TLB

- All that must be in memory for a process to run fairly well is the Working Set of Pages. This is Spatial Locality.

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

- These are extra slides that used to be included in lecture notes, but have been moved to this, the “bonus” area to serve as a supplement.
- The slides will appear in the order they would have in the normal presentation.

Address Map, Mathematically

\[ V = \{0, 1, \ldots, n - 1\} \] virtual address space \((n > m)\)
\[ M = \{0, 1, \ldots, m - 1\} \] physical address space
\[ \text{MAP: } V \rightarrow M \cup \{e\} \] address mapping function

\[ \text{MAP}(a) = a' \text{ if data at virtual address } a \text{ is present in physical address } a' \text{ and } a' \text{ in } M \]
\[ = e \text{ if data at virtual address } a \text{ is not present in } M \]

Processor
\[ \text{Name Space } V \]
\[ \text{Addr Trans} \]
\[ \text{Mechanism} \]
\[ \text{OS fault handler} \]
\[ \text{Main Memory} \]
\[ \text{Disk} \]

physical address
OS performs this transfer
page fault
0