Virtual Memory III

Instructor: Dan Garcia

Virtual Memory Mapping Function

- How large is main memory? Disk?
  - Don’t know! Designed to be interchangeable components
  - Need a system that works regardless of sizes
- Use lookup table (page table) to deal with arbitrary mapping
  - Index lookup table by # of pages in VM
  - Size of PM will affect size of stored translation

Address Mapping

- Pages are aligned in memory
  - Border address of each page has same lowest bits
  - Page size is same in VM and PM, so denote lowest $O = \log_2(\text{page size/byte})$ bits as page offset
- Use remaining upper address bits in mapping
  - Tells you which page you want (similar to Tag)

Address Mapping: Page Table

- Page Table functionality:
  - Incoming request is Virtual Address (VA), want Physical Address (PA)
  - Physical Offset = Virtual Offset (page-aligned)
  - So just swap Virtual Page Number (VPN) for Physical Page Number (PPN)

• Implementation?
  - Use VPN as index into PT
  - Store PPN and management bits (Valid, Access Rights)
  - Does NOT store actual data (the data sits in PM)

Page Table Layout

Virtual Address:

- 16 KiB pages
- 40-bit virtual addresses
- 64 GiB physical memory

Question: How many bits wide are the following fields?

- A) 26 26
- B) 24 20
- C) 22 22
- D) 26 22
Retrieving Data from Memory

1) Access page table for address translation

2) Access correct physical address

Requires two accesses of physical memory!

Where Are TLBs Located?

- Which should we check first: Cache or TLB?
  - Can cache hold requested data if corresponding page is not in physical memory? **No**
  - With TLB first, does cache receive VA or PA?

Address Translation Using TLB

- **TLBs** usually small, typically 16 – 512 entries
- **TLB access time** comparable to cache (≈ main memory)
- **TLBs can have associativity**
  - Usually fully/highly associative

Question: How many bits wide are the following?
- 16 KiB pages
- 40-bit virtual addresses
- 64 GiB physical memory
- 2-way set associative TLB with 512 entries

<table>
<thead>
<tr>
<th>Valid</th>
<th>Dirty</th>
<th>Ref</th>
<th>Access Rights</th>
<th>TLB Tag</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>XX</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TLB Tag</th>
<th>TLB Index</th>
<th>TLB Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>A) 12</td>
<td>14</td>
<td>38</td>
</tr>
<tr>
<td>B) 18</td>
<td>8</td>
<td>45</td>
</tr>
<tr>
<td>C) 14</td>
<td>12</td>
<td>40</td>
</tr>
<tr>
<td>D) 17</td>
<td>9</td>
<td>43</td>
</tr>
</tbody>
</table>

Administrivia

- **TAs?**
Fetching Data on a Memory Read

1) Check TLB (input: VPN, output: PPN)
   - **TLB Hit**: Fetch translation, return PPN
   - **TLB Miss**: Check page table (in memory)
     - **Page Table Hit**: Load page table entry into TLB
     - **Page Table Miss (Page Fault)**: Fetch page from disk to memory, update corresponding page table entry, then load entry into TLB

2) Check cache (input: PPN, output: data)
   - **Cache Hit**: Return data value to processor
   - **Cache Miss**: Fetch data value from memory, store it in cache, return it to processor

Page Faults

- Load the page off the disk into a free page of memory
  - Switch to some other process while we wait
- Interrupt thrown when page loaded and the process’ page table is updated
  - When we switch back to the task, the desired data will be in memory
- If memory full, replace page (LRU), writing back if necessary, and update both page table entries
  - Continuous swapping between disk and memory called “thrashing”

Performance Metrics

- VM performance also uses Hit/Miss Rates and Miss Penalties
  - **TLB Miss Rate**: Fraction of TLB accesses that result in a TLB Miss
  - **Page Table Miss Rate**: Fraction of PT accesses that result in a page fault
- Caching performance definitions remain the same
  - Somewhat independent, as TLB will always pass PA to cache regardless of TLB hit or miss

Data Fetch Scenarios

- Are the following scenarios for a single data access possible?
  - TLB Miss, Page Fault: Yes
  - TLB Hit, Page Table Hit: No
  - TLB Miss, Cache Hit: Yes
  - Page Table Hit, Cache Miss: Yes
  - Page Fault, Cache Hit: No

Question: A program tries to load a word at X that causes a TLB miss but not a page fault. Are the following statements TRUE or FALSE?

1) The page table does not contain a valid mapping for the virtual page corresponding to the address X
2) The word that the program is trying to load is present in physical memory

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>F</td>
</tr>
<tr>
<td>B</td>
<td>F</td>
</tr>
<tr>
<td>C</td>
<td>T</td>
</tr>
<tr>
<td>D</td>
<td>T</td>
</tr>
</tbody>
</table>

VM Performance

- Virtual Memory is the level of the memory hierarchy that sits *below* main memory
  - TLB comes *before* cache, but affects transfer of data from disk to main memory
  - Previously we assumed main memory was lowest level, now we just have to account for disk accesses
- Same CPI, AMAT equations apply, but now treat main memory like a mid-level cache
Typical Performance Stats

Caching
- cache entry
- cache block (=32 bytes)
- cache miss rate (1% to 20%)
- cache miss (=100 cycles)

Demand paging
- page frame
- page frame size (=4Ki bytes)
- page hit rate (<0.001%)
- page miss rate (≈5M cycles)

Impact of Paging on AMAT (1/2)

- Memory Parameters:
  - L1 cache hit = 1 clock cycles, hit 95% of accesses
  - L2 cache hit = 10 clock cycles, hit 60% of L1 misses
  - DRAM = 200 clock cycles (=100 nanoseconds)
  - Disk = 20,000,000 clock cycles (=10 milliseconds)

- Average Memory Access Time (no paging):
  - 1 + 5%×10 + 5%×40%×200 = 5.5 clock cycles

- Average Memory Access Time (with paging):
  - 5.5 (AMAT with no paging) + ?

Impact of Paging on AMAT (2/2)

- Average Memory Access Time (with paging) =
  - 5.5 + 5%×40%×(1-HR_{Mem})×20,000,000

- AMAT if HR_{Mem} = 99%?
  - 5.5 + 0.02×0.01×20,000,000 = 405.5

- AMAT if HR_{Mem} = 99.9999%?
  - 5.5 + 0.02×0.000001×20,000,000 = 5.9

Impact of TLBs on Performance

- Each TLB miss to Page Table ~ L1 Cache miss

- TLB Reach: Amount of virtual address space that can be simultaneously mapped by TLB:
  - TLB typically has 128 entries of page size 4-8 KiB
  - 128 × 4 KiB = 512 KiB = just 0.5 MiB

- What can you do to have better performance?
  - Multi-level TLBs
  - Variable page size (segments)
  - Special situationally-used "superpages"

Aside: Context Switching

- How does a single processor run many programs at once?
- Context switch: Changing of internal state of processor (switching between processes)
  - Save register values (and PC) and change value in Page Table Base register
- What happens to the TLB?
  - Current entries are for different process
  - Set all entries to invalid on context switch

Virtual Memory Summary

- User program view:
  - Contiguous memory
  - Start from some set VA
  - "infinitely" large
  - Is the only running program
- Reality:
  - Non-contiguous memory
  - Start wherever available memory is
  - Finite size
  - Many programs running simultaneously
- Virtual memory provides:
  - Illusion of contiguous memory
  - All programs starting at same set address
  - Illusion of "infinite memory" (2^20 or 2^32 bytes)
  - Protection, Sharing
- Implementation:
  - Divide memory into chunks (pages)
  - OS controls page table that maps virtual into physical addresses
  - memory as a cache for disk
  - TLB is a cache for the page table