Overview

° Translation Lookaside Buffer (TLB) Mechanism
° Two level page Table

Three Advantages of Virtual Memory (#1/2)

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

Three Advantages of Virtual Memory (#2/2)

2) Protection:
   • Different processes protected from each other
   • Different pages can be given special behavior
     - (Read Only, Invisible to user programs, etc).
   • Privileged data protected from User programs
   • Very important for protection from malicious programs ⇒ Far more “viruses” under Microsoft Windows

3) Sharing:
   • Can map same physical page to multiple users ("Shared memory")
Why Translation Lookaside Buffer (TLB)?

- Every paged virtual memory access must be checked against Entry of Page Table in memory to provide VA \(\rightarrow\) PA translation and protection.
- Cache of Page Table Entries makes address translation possible without memory access in common case to make it fast.

Recall: Typical TLB Format

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Physical Address</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).
  - **Ref**: Used to help calculate LRU on replacement.
  - **Dirty**: since use write back, need to know whether or not to write page to disk when replaced.

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.
- ARM follows Option 1: Hardware does the loading of new Page Table Entry.

TLB Miss

- If the address is not in the TLB, ARM’s Translation Table Walk Hardware is invoked to retrieve the relevant entry from translation table held in main memory.
- There are two possibilities:

<table>
<thead>
<tr>
<th>valid</th>
<th>virtual</th>
<th>physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>9</td>
</tr>
</tbody>
</table>
TLB Miss (If the Data is in Memory)

- Translation Table Walk Hardware simply adds the entry to the TLB, evicting an old entry from the TLB if no empty slot.
- Fetch Translation once on TLB.

<table>
<thead>
<tr>
<th>valid</th>
<th>virtual</th>
<th>physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>9</td>
</tr>
</tbody>
</table>

TLB Miss (if the Data is on Disk)

- A Page Fault (Abort exception) is issued to the processor.
- The OS loads the page off the disk into a free block of memory, using a DMA transfer.
  - Meantime OS switches to some other process waiting to be run.
- When the DMA is complete, Processor gets an interrupt and OS update the process's page table and TLB.
  - So when OS switches back to the task, the desired data will be in memory.

What if We Don't Have Enough Memory?

- OS chooses some other page belonging to a program and transfer it onto the disk if it is dirty.
  - If clean (other copy is up-to-date), just overwrite that data in memory.
  - OS chooses 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 on disk.

Translation Look-Aside Buffers

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

- Let’s say we’re fetching some data:
  - Check TLB (input: VPN, output: PPN)
    - hit: fetch translation
    - miss: check pagetable (in memory)
      - pagetable hit: fetch translation
      - pagetable 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

Virtual Memory Problem #3

- Page Table too big!
  - 4GB Virtual Memory ÷ 4 KB page
    ⇒ ~ 1 million Page Table Entries
    ⇒ 4 MB just for Page Table for 1 process,
    25 processes ⇒ 100 MB for Page Tables!

- Variety of solutions to trade off memory size of mapping function for slower when miss TLB
  - Make TLB large enough, highly associative so rarely miss on address translation
  - COMP3231: Operating Systems, will go over more options and in greater depth

2-level Page Table

ARM MMU uses 2-level page Table

Virtual Memory

2nd Level Page Tables

Super Page Table

Physical Memory

64 MB

Stack

Heap

Static

Code

Page Table Shrink:

- Single Page Table
  - Page Number Offset
    - 20 bits 12 bits
    - $2^{20} = 4$MB 1st level page Table per process!

- Multilevel Page Table
  - Super Page No. Page Number Offset
    - 10 bits 10 bits 12 bits
    - $2^{10} \times 2^{20} = 2^{20} = 4$MB 2nd level page Table per process!

- But: Only have second level page table for valid entries of super level page table
Space Savings for Multi-Level Page Table

If only 10% of entries of Super Page Table have valid entries, then total mapping size is roughly 1/10-th of single level page table.

Reading Material


Sendo X Smart Phone

On the 21st October Sendo announced its new highly featured multimedia smartphone, the Sendo X.

The Sendo X sets itself apart from other multimedia smartphones with a number of innovations, specifically in the area of video, audio, camera, connectivity and internet functionality.

Video: The Sendo X features a 176x220 TFT display with up to 65,536 colours. It has a built-in camcorder function with more than half an hour of recording of quality video and audio.

Camera: The VGA still/video camera is highly specified. It offers 4 x digital zoom and an integrated flash with automatic red-eye reduction.

Audio: The Sendo X has been designed to offer best in class performance, with a 64 voice polyphonic capability.

The Sendo X is the first Sendo smartphone based on Symbian OS and the Series 60 user interface.

Address Translation & 3 Exercises

Virtual Address

Physical Address

What are the trade off in designing such a system?
Address Translation Exercise 1 (#1/2)
Exercise:
° 40-bit VA, 16 KB pages, 36-bit PA
° Number of bits in Virtual Page Number?
  a) 18; b) 20; c) 22; d) 24; e) 26; f) 28
° Number of bits in Page Offset?
  a) 8; b) 10; c) 12; d) 14; e) 16; f) 18
° Number of bits in Physical Page Number?
  a) 18; b) 20; c) 22; d) 24; e) 26; f) 28

Address Translation Exercise 1 (#2/2)
° 40-bit virtual address, 16 KB (2^14 B)
  Virtual Page Number (26 bits)  Page Offset (14 bits)
° 36-bit virtual address, 16 KB (2^14 B)
  Physical Page Number (22 bits)  Page Offset (14 bits)

Address Translation Exercise 2 (#1/2)
Exercise:
° 40-bit VA, 16 KB pages, 36-bit PA
° 2-way set-assoc TLB: 256 "slots", 2 per slot
° Number of bits in TLB Index?
  a) 8; b) 10; c) 12; d) 14; e) 16; f) 18
° Number of bits in TLB Tag?
  a) 18; b) 20; c) 22; d) 24; e) 26; f) 28
° Approximate Number of bits in TLB Entry?
  a) 32; b) 36; c) 40; d) 42; e) 44; f) 46

Address Translation Exercise 2 (#2/2)
° 2-way set-assoc data cache, 256 (2^8) "slots", 2 TLB entries per slot => 8 bit index
  TLB Tag (18 bits)  TLB Index (8 bits)  Page Offset (14 bits)
  Virtual Page Number (26 bits)
° Data Cache Entry: Valid bit, Dirty bit, Access Control (2-3 bits?), Virtual Page Number, Physical Page Number
  V  D  Access (3 bits)  TLB Tag (18 bits)  Physical Page No. (22 bits)
Address Translation Exercise 3 (#1/2)

- Exercise:
  - 40-bit VA, 16 KB pages, 36-bit PA
  - 2-way set-associative TLB: 256 "slots", 2 per slot
  - 64 KB data cache, 64 Byte blocks, 2 way S.A.

- Number of bits in Cache Offset?
  a) 6; b) 8; c) 10; d) 12; e) 14; f) 16

- Number of bits in Cache Index?
  a) 6; b) 9; c) 10; d) 12; e) 14; f) 16

- Number of bits in Cache Tag?
  a) 18; b) 20; c) 21; d) 24; e) 26; f) 28

- Approximate No. of bits in Cache Entry?

Address Translation 3 (#2/2)

- 2-way set-assoc data cache, 64K/64 = 1K (2^10) blocks, 2 entries per slot => 512 slots => 10 bit index

- Data Cache Entry: Valid bit, Dirty bit, Cache tag + 64 Bytes of Data

Things to Remember

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

- TLB to reduce performance cost of VM

- Need more compact representation to reduce memory size cost of simple 1-level page table (especially 32 ⇒ 64-bit address)