Lecture 19:
Virtual memory - practice

Computer Architecture and Systems Programming
(252-0061-00)

Timothy Roscoe
Herbstsemester 2012

Last Time: Address Translation

Virtual address

<table>
<thead>
<tr>
<th>Virtual page number (VPN)</th>
<th>Virtual page offset (VPO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page table base register (PTBR)</td>
<td></td>
</tr>
<tr>
<td>Page table address for process</td>
<td></td>
</tr>
<tr>
<td>Valid bit = 0: page not in memory (page fault)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Physical page number (PPN)</th>
<th>Physical page offset (PPO)</th>
</tr>
</thead>
</table>

Physical address

Last Time: Page Fault

A TLB hit eliminates a memory access

TLB Miss

A TLB miss incurs an add’l memory access (the PTE)
Fortunately, TLB misses are rare

Today

- A note on Terminology
- Virtual memory (VM)
  - Multi-level page tables
- Case study: VM system on P6
- Historical aside: VM system on the VAX
- x86-64 and 64-bit paging
- Performance optimization for VM system
**Terminology**

- **Virtual Page** may refer to:
  - Page-aligned region of virtual address space
  - Contents thereof (different – might appear on disk)
- **Physical Page**:
  - Page-aligned region of physical memory (RAM)
- **Physical Frame** (=Physical Page)
  - Alternative terminology
  - Page = contents, Frame = container
  - Page size may be ≠ frame size (rarely)

**Today**

- A note on Terminology
- **Virtual memory (VM)**
  - Multi-level page tables
- Case study: VM system on P6
- Historical aside: VM system on the VAX
- x86-64 and 64-bit paging
- Performance optimization for VM system

**Multi-Level Page Tables**

- Given:
  - 4KB ($2^{12}$) page size
  - 48-bit address space
  - 4-byte PTE
- Problem:
  - Would need a 256 GB page table!  
    - $2^{48} \times 2^{-12} \times 2^2 = 2^{38}$ bytes
- Common solution
  - Multi-level page tables
  - Example: 2-level page table
  - Level 1 table: each PTE points to a page table
  - Level 2 table: each PTE points to a page
    (paged in and out like other data)
  - Level 1 table stays in memory
  - Level 2 tables paged in and out

**2-Level Page Table Hierarchy**

- Level 1 page table
- Level 2 page tables
- Virtual memory

**Translating with a k-level Page Table**

- Virtual Address
- Physical Address

**Today**

- A note on Terminology
- **Virtual memory (VM)**
  - Multi-level page tables
- Case study: VM system on P6
- Historical aside: VM system on the VAX
- x86-64 and 64-bit paging
- Performance optimization for VM system
Intel P6

- Internal designation for successor to Pentium
  - Which had internal designation P5
- Fundamentally different from Pentium
  - Out-of-order, superscalar operation
- Resulting processors
  - Pentium Pro (1996)
  - Pentium II (1997)
  - L2 cache on same chip
  - Pentium III (1999)
- Different microarchitecture to the Pentium 4
  - Similar memory system
  - P4 abandoned by Intel in 2005 for P6-based Core 2 Duo

P6 Memory System

- 32 bit address space
- 4 KB page size
- L1, L2, and TLBs
  - 4-way set associative
  - Inst TLB
    - 32 entries
    - 8 sets
  - Data TLB
    - 64 entries
    - 16 sets
  - L1 i-cache and d-cache
    - 16 KB
    - 32 B line size
    - 128 sets
  - L2 cache
    - unified
    - 128 KB–2 MB

Review of Abbreviations

- Components of the virtual address (VA)
  - TLBI: TLB index
  - TLBT: TLB tag
  - VPO: virtual page offset
  - VPN: virtual page number
- Components of the physical address (PA)
  - PPO: physical page offset (same as VPO)
  - PPN: physical page number
  - CO: byte offset within cache line
  - CI: cache index
  - CT: cache tag

P6 2-level Page Table Structure

- Page directory
  - 1024 4-byte page directory entries (PDEs) that point to page tables
  - One page directory per process
  - Page directory must be in memory when its process is running
  - Always pointed to by PDDBR
  - Large page support:
    - Make PD the page table
    - Fixes page size to 4MB (why?)
- Page tables
  - 1024 4-byte page table entries (PTEs) that point to pages
  - Size: exactly one page
  - Page tables can be paged in and out

P6 Page Directory Entry (PDE)

- Page table physical base address: 20 most significant bits of physical page table address (forces page tables to be 4KB aligned)
- Avail: These bits available for system programmers
- G: global page (don’t evict from TLB on task switch)
- PS: page size 4K (0) or 4M (1)
- A: accessed (set by MMU on reads and writes, cleared by software)
- CD: cache disabled (1) or enabled (0)
- WT: write-through or write-back cache policy for this page table
- U/S: user or supervisor mode access
- R/W: read-only or read-write access
- P: page table is present in memory (1) or not (0)
**P6 Page Table Entry (PTE)**

<table>
<thead>
<tr>
<th></th>
<th>31</th>
<th>12</th>
<th>11</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>avail</td>
<td>0</td>
<td>G</td>
<td>0</td>
<td>D</td>
<td>A</td>
<td>CD</td>
<td>WT</td>
<td>U/S</td>
<td>R/W</td>
<td>P=0</td>
<td>PTE</td>
<td>TLBI</td>
<td>TLBT</td>
</tr>
</tbody>
</table>

- **Page base address**: 20 most significant bits of physical page address (forces pages to be 4 KB aligned)
- **Avail**: available for system programmers
- **G**: global page (don’t evict from TLB on task switch)
- **D**: dirty (set by MMU on writes)
- **A**: accessed (shared by MMU on reads and writes)
- **CD**: cache disabled or enabled
- **WT**: write-through or write-back cache policy for this page
- **U/S**: user/supervisor
- **R/W**: read/write
- **P**: page is present in physical memory (1) or not (0)

---

**Representation of VM Addr. Space**

- **Simplified Example**
  - 16 page virtual address space
- **Flags**
  - **P**: Is entry in physical memory?
  - **M**: Has this part of VA space been mapped?

---

**P6 TLB Translation**

- **TLB entry (not all documented, so this is speculative):**
  - **V**: indicates a valid (1) or invalid (0) TLB entry
  - **TLBTag**: disambiguates entries cached in the same set
  - **PPN**: translation of the address indicated by index & tag
  - **G**: page is “global” according to PDE, PTE
  - **S**: page is “supervisor-only” according to PDE, PTE
  - **W**: page is writable according to PDE, PTE
  - **D**: PTE has already been marked “dirty” (once is enough)

- **Structure of the data TLB**
  - 16 sets, 4 entries/set

---

**Translating with the P6 TLB**

1. **Partition VPN into TLBT and TLBI**.
2. **Is the PTE for VPN cached in set TLBI**?
3. **Yes**: Check permissions, build physical address
4. **No**: Read PTE (and PDE if not cached) from memory and build physical address
Translating with P6 Page Tables (case 1/1)

- Page table and page present
- MMU Action:
  - MMU builds physical address and fetches data word
- OS action
  - None

Translating with P6 Page Tables (case 1/0)

- Page table present, page missing
- MMU Action:
  - Page fault exception
  - Handler receives the following args:
    - %eip that caused fault
    - VA that caused fault
    - Fault caused by non-present page or page-level protection violation
      - Read/write
      - User/supervisor

Translating with P6 Page Tables (case 1/0, cont.)

- OS Action:
  - Check for a legal virtual address.
  - Read PTE through PDE.
  - Find free physical page (swapping out current page if necessary)
  - Read virtual page from disk into physical page
  - Adjust PTE to point to physical page, set p=1
  - Restart faulting instruction by returning from exception handler

Translating with P6 Page Tables (case 0/1)

- Page table missing, page present
- Introduces consistency issue
  - Potentially every page-out requires update of disk page table
- Linux disallows this
  - If a page table is swapped out, then swap out its data pages too

Translating with P6 Page Tables (case 0/0)

- Page table and page missing
- MMU Action:
  - Page fault

Translating with P6 Page Tables (case 0/0, cont.)

- OS action:
  - Swap in page table
  - Restart faulting instruction by returning from handler
- Like case 0/1 from here on.
  - Two disk reads
P6 L1 Cache Access

- CPU
- TLB
- L1 (128 sets, 4 lines/set)
- L2 and DRAM

- Virtual address (VA)
- Physical address (PA)

- VPN
- VPO
- PDE
- PTE
- PDBR
- PPN
- PPO

- TLB (16 sets, 4 entries/set)
- TLBT

- VPN1
- VPN2

- L1 hit
- L1 miss

- CT
- CO
- CI

- Speeding Up L1 Access

  - Observation
    - Bits that determine CI identical in virtual and physical address
    - Can index into cache while address translation taking place
    - Generally we hit in TLB, so PPN bits (CT bits) available next
    - "Virtually indexed, physically tagged"
    - Cache carefully sized to make this possible

Today

- A note on Terminology
- Virtual memory (VM)
  - Multi-level page tables
- Case study: VM system on P6
- Historical aside: VM system on the VAX
- x86-64 and 64-bit paging
- Performance optimization for VM system

Historical aside: virtual page tables

- Same problem: linear page table can be large.

On the VAX:

- Page size = 512 bytes (so offset = 9 bits)
- Virtual address space = 32 bits
  ⇒ page table index = 23 bits
Solution: put the linear table into virtual memory!

- Of course, most of the PTEs are not used
  - Invalid translation: saves space
- TLB hides most of the double lookups

VAX translation process

• Not so bizarre after all:
  This really is a 2-level page table:

Today

- A note on Terminology
- Virtual memory (VM)
  - Multi-level page tables
- Case study: VM system on P6
- Historical aside: VM system on the VAX
- x86-64 and 64-bit paging
- Performance optimization for VM system
x86-64 Paging

- Origin
  - AMD’s way of extending x86 to 64-bit instruction set
  - Intel has followed with “EM64T”
- Requirements
  - 48-bit virtual address
    - 256 terabytes (TB)
    - Not yet ready for full 64 bits
      - Nobody can buy that much DRAM yet
      - Multi-level array map may not be the right data structure
  - 52-bit physical address = 40 bits for PPN
    - Requires 64-bit table entries
    - Keep traditional x86 4KB page size, and same size for page tables
      - (4096 bytes per PT) / (8 bytes per PTE) = only 512 entries per page

Today

- A note on Terminology
- Virtual memory (VM)
  - Multi-level page tables
- Case study: VM system on P6
- Historical aside: VM system on the VAX
- x86-64 and 64-bit paging
- Performance optimization for VM system

Large Pages

- 4MB on 32-bit, 2MB on 64-bit
- Simplify address translation
- Useful for programs with very large, contiguous working sets
  - Reduces compulsory TLB misses
- How to use (Linux)
  - hugetlbfs support (since at least 2.6.16)
  - Use l1bhtulbfs
    - \{m,c,re\}alloc replacements

Buffering: Example MMM

- Blocked for cache
- Assume blocking for L2 cache
  - say, 512 MB = 2^{19} B = 2^{16} doubles = C
  - 3B^2 < C means B = 150

Buffering: (cont.)

- But: Look at one iteration
  - assume > 4 KB = 512 doubles
  - each row used O(\text{B}) times
    - but every time O(\text{B^2}) ops between
- Consequence
  - Each row is on different page
  - More rows than TLB entries: TLB thrashing
    - Solution: buffering = copy block to contiguous memory
    - O(\text{B}) cost for O(\text{B^2}) operations
Next time: I/O Devices

- What is a device?
- Registers
  - Example: NS16550 UART
- Interrupts
- Direct Memory Access (DMA)
- PCI (Peripheral Component Interconnect)