Virtual Memory: System

CSCI 2021: Machine Architecture and Organization

Pen-Chung Yew
Department Computer Science and Engineering
University of Minnesota

With Slides from Bryant, O’Hallaron and Zhai

Outline

• Simple memory system example
  • Case study: Core i7/Linux memory system
• Memory mapping

Address Translation: Page Hit

1) Processor sends virtual address to MMU
2-3) MMU fetches PTE from page table in cache/memory
4) MMU sends physical address to cache/memory
5) Cache/memory sends data word to processor

Total of 2 trips to cache/memory per memory operation!!

A Simple Memory System (Page Hit)

Address Translation: Page Fault (Miss)

1) Processor sends virtual address to MMU
2-3) MMU fetches PTE from page table in cache/memory
4) Valid bit is zero, so MMU triggers page fault exception
5) Exception Handler in OS identifies a victim (and, if dirty, pages it out to disk)
6) Exception Handler pages in new page and updates PTE in memory
7) Handler returns to original process, restarting faulting instruction

Review: Address Translation Symbols

• Basic Parameters
  • \( N = 2^n \): Number of addresses in virtual address space
  • \( M = 2^p \): Number of addresses in physical address space
  • \( P = 2^b \): Page size (bytes)

• Components of virtual address (VA)
  • VPO: Virtual page offset
  • VPN: Virtual page number
  • TLBI: TLB index
  • TLBT: TLB tag

• Components of physical address (PA)
  • PPO: Physical page offset (same as VPO)
  • PPN: Physical page number

• Component of a cache address (PA)
  • CO: Byte offset within cache line
  • CI: Cache index
  • CT: Cache tag
Simple Memory System: TLB

- TLB - a small cache for address translation
  - 16 entries
  - 4-way associative
  - 4 sets (16/4 = 4 = 2^2) total in TLB

<table>
<thead>
<tr>
<th>Virtual address</th>
<th>TLBI = TLB Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPN</td>
<td>VPO</td>
</tr>
</tbody>
</table>

TLB

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>VPN</th>
<th>VPO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Simple Memory System: L1 Cache

- Physically addressed, Direct mapped
- 4-byte block size ➔ 2^4 cache block size
- 16 lines ➔ 2^4 lines;

**L1 Cache**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Valid</th>
<th>CT</th>
<th>CO</th>
<th>PPO</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Y</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>Y</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Physical address:

- 4/6/16

Address Translation Example #2

**Virtual Address: 0x03D4**

<table>
<thead>
<tr>
<th>TLBI = TLB Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Physical Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x03D4</td>
<td>CT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag</th>
<th>Valid</th>
<th>CT</th>
<th>CO</th>
<th>PPO</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Address Translation Example #3

**Virtual Address: 0x0B8F**

<table>
<thead>
<tr>
<th>TLBI = TLB Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Physical Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0B8F</td>
<td>CT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag</th>
<th>Valid</th>
<th>CT</th>
<th>CO</th>
<th>PPO</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Outline

- Simple memory system example
- Case study: Core i7/Linux memory system
- Memory mapping

Review: Multi-Level Page Tables

- Suppose:
  - 4KB (2^{12}) page size, 48-bit address space, 4-byte PTE

- Problem:
  - Would need a 256 GB page table!
  - \( 2^{40}/2^{12} \times 2^{2} = 2^{38} \) bytes (~256GB)

- Common solution:
  - Multi-level page tables
  - Example: 2-level page table
    - Level 1 table: each PTE points to a page table (always memory resident)
    - Level 2 table: each PTE points to a page of Level-2 page tables (paged in and out like any other data)

Review: A Two-Level Page Table Hierarchy

End-to-end Core i7 Address Translation

Core i7 Page Table Translation
Each entry references a 4KB child page table:
- **P**: Child page table present in physical memory (1) or not (0).
- **R/W**: Read-only or read-write access permission for all reachable pages.
- **U/S**: User or supervisor (kernel) mode access permission for all reachable pages.
- **WT**: Write-through or write-back cache policy for the child page table.
- **CD**: Caching disabled or enabled for the child page table.
- **A**: Reference bit (set by MMU on reads and writes, cleared by software).
- **PS**: Page size either 4 KB or 4 MB (defined for Level 1 PTEs only).
- **XD**: Global page (don't evict from TLB on task switch).

Page table physical base address: 40 most significant bits of physical page table address (forces page tables to be 4KB aligned)

**Observation**
- Bits that determine CI identical in virtual and physical address
- Can index into cache while address translation (TLB lookup) 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!!

**Virtual Memory of a Linux Process**

**Kernel virtual memory**
- Process-specific data structures (kernel, task and mm structs, kernel stacks)

**User virtual memory**
- User stack
- Memory mapped region for shared libraries
- Runtime heap (mallocs)
- Uninitialized data (.bss)
- Initialized data (.data)
- Program text (.text)

**Process code and data**

**Program with Linux Process**

---

**Core i7 Level 1-3 Page Table Entries**

<table>
<thead>
<tr>
<th>L1</th>
<th>L2</th>
<th>L3</th>
<th>L4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Available for OS (page table location on disk)

**Core i7 Level 4 Page Table Entries**

<table>
<thead>
<tr>
<th>L1</th>
<th>L2</th>
<th>L3</th>
<th>L4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Available for OS (page table location on disk)

---

**Cute Trick for Speeding Up L1 Access**

- Physical address (PA)
- Virtual address (VA)
- Tag Check
- L1 Cache
- Physical Address (PA)
- Virtual Address (VA)
- Page Table Entries (PTE)
- Physical Pages (PPN)
- Page Table (PT)

**Review: Core i7 Page Table Translation**

**Review: Core i7 Address Translation**

**Physical Address (PA)**

---

**Virtual Memory**

<table>
<thead>
<tr>
<th>Virtual address</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPN 1</td>
</tr>
<tr>
<td>VPN 2</td>
</tr>
<tr>
<td>VPN 3</td>
</tr>
<tr>
<td>VPN 4</td>
</tr>
<tr>
<td>VPN 5</td>
</tr>
<tr>
<td>VPN 6</td>
</tr>
</tbody>
</table>

**Physical Address**

---

**Different for each process**

**Identical for each process**

---

**Kernel virtual memory**

**User virtual memory**

---

**Virtual Memory of a Linux Process**

---

**Process code and data**

---
VM as Collection of “Areas”

- `pgd`:
  - Page global directory address
  - Points to L1 page table
- `vm_prot`:
  - Read/write permissions for this area
- `vm_flags`:
  - Pages shared with other processes or private to this process

Outline

- Simple memory system example
- Case study: Core i7/Linux memory system
- Memory mapping

Demand paging

- **Key point**: no virtual pages are copied into physical memory until they are referenced!
  - Known as *demand paging*
- Crucial for time and space efficiency

Sharing: *Shared Objects*

- Process 1 maps a shared object.

Sharing: *Private* Copy-on-write (COW) Objects

- Two processes mapping a *private* copy-on-write (COW) object.
- Area flagged as *private* copy-on-write
- PTEs in private areas are flagged as *read-only*
Sharing: *Private* Copy-on-write (COW) Objects

- Instruction **writing** to private page triggers protection fault.
- Handler creates new R/W page.
- Instruction restarts upon handler return.
- Copying deferred as long as possible!