Virtual Memory: Concepts
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
- Address spaces
  - VM as a tool for caching
  - VM as a tool for memory management
  - VM as a tool for memory protection
  - Address translation

A System Using Physical Addressing
- Used in "simple" systems like embedded microcontrollers in devices like cars, elevators, and digital picture frames

A System Using Virtual Addressing
- Used in all modern servers, desktops, and laptops
- One of the great ideas in computer science

Address Spaces
- Linear address space: Ordered set of contiguous non-negative integer addresses. Each address stores one byte of information (data/instruction). (0, 1, 2, 3 …)
- Virtual address space: Set of N = 2^n virtual addresses (n = 32 for 32-bit ISA) [0, 1, 2, 3, …, N-1]
- Physical address space: Set of M = 2^m physical addresses (m = 24 for 16MB) [0, 1, 2, 3, …, M-1]
- Clean distinction between data (bytes) and their attributes (addresses)
- Each object can have multiple addresses for each level of memory hierarchy

Every byte in main memory has one physical address, and one (or more) virtual addresses
Address Calculation Table: # of addr bits

<table>
<thead>
<tr>
<th>2^2</th>
<th>2^3</th>
<th>2^4</th>
<th>2^5</th>
<th>2^6</th>
<th>2^7</th>
<th>2^8</th>
<th>2^9</th>
<th>2^10</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td>256</td>
<td>512</td>
<td>1024</td>
</tr>
</tbody>
</table>

Why Virtual Memory (VM)?

- Uses main memory efficiently
  - Use DRAM as a cache for the parts of a virtual address space
- Simplifies memory management
  - Each process gets the same uniform linear address space
- Isolates address spaces (for protection)
  - One process can’t access or interfere with another’s memory
  - User program cannot access privileged kernel information

Outline

- Address spaces
- VM as a tool for caching
- VM as a tool for memory management
- VM as a tool for memory protection
- Address translation

VM as a Tool for Caching

- Virtual memory is an array of N contiguous bytes stored on disk.
- The contents of the array on disk are cached in physical main memory (DRAM cache)
  - These cache blocks are called pages (size is P = 2^b bytes)

DRAM Main Memory Organization

- DRAM cache (main memory) organization driven by the enormous miss penalty
  - DRAM is about 100X slower than SRAM
  - Disk is about 100,000X slower than DRAM
- Consequences in design
  - Large page (block) size: typically 4-8 KB, sometimes 4 MB
  - Fully associative
    - Any Virtual Page can be placed in any Physical Page
    - Requires a “large” mapping function – different from CPU caches
  - Highly sophisticated, expensive replacement algorithms
    - Too complicated and open-ended to be implemented in hardware
    - Software (i.e., operating system) does it, with hardware support
  - Write-Back rather than Write-Through

DRAM Main Memory Organization (cont.)

- Consequences
  - Large page size – amortize large miss penalty (to disk)
  - Fully associative – completely get rid of conflict misses
    - Need to know where pages are residing in main memory
    - Too expensive to use hardware for concurrent comparison of tags as in CPU cache memories → use software to do it (with some help from hardware)
- Solution – Use a page table for table lookup
  - One page table entry per virtual memory page
  - Each entry stores the location of the page in main memory
  - Too slow → use caching (Translation Lookaside Buffer (TLB)) for rescue
Page Tables

- A page table is an array of page table entries (PTEs) that keep the locations of pages in main memory, i.e., maps virtual pages to physical pages.
- Per-process kernel data structure in DRAM

For example: $2^{12}/2^{12} = 2^0$ PTEs

Page Fault (Miss)

- Page fault (miss): reference to VM word (in VP3) that is not in physical memory (DRAM cache miss)

Page Hit

- Page hit: reference to VM word (in VP2) that is in physical memory (DRAM cache hit)

Handling Page Fault

- Page miss causes page fault (an exception)
- Page fault handler (in OS) selects a victim to be evicted (here VP 4) to disk
- VP3 is brought into main memory (DRAM)

Handling Page Fault

- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here VP 4) to disk
- VP3 is brought into main memory (DRAM)
Handling Page Fault

- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here VP 4) to disk
- Offending instruction is restarted: page hit!

Locality to the Rescue Again!

- Virtual memory works because of locality
- At any point in time, programs tend to access a set of active virtual pages called the working set
  - Programs with better temporal locality will have smaller working sets
- If (working set size < main memory size)
  - Good performance for one process after compulsory misses
- If (SUM(working set sizes) > main memory size)
  - Thrashing: Performance meltdown where pages are swapped (copied) in and out continuously between Main Memory (DRAM) and Disks

Outline

- Address spaces
- VM as a tool for caching
- VM as a tool for memory management
- VM as a tool for memory protection
- Address translation

VM as a Tool for Memory Management

- Key idea: each process has its own virtual address space
  - It can view memory as a simple linear array
  - Mapping function scatters addresses through physical memory
    - Well chosen mappings simplify memory allocation and management
- Memory allocation
  - Each virtual page can be mapped to any physical page
  - A virtual page can be stored in different physical pages at different times
- Sharing code and data among processes
  - Map virtual pages to the same physical page (here: PP 6)
VM for Memory Protection

- Extend PTEs with permission bits
- Page fault handler checks these before remapping
  - If violated, send process SIGSEGV (segmentation fault)

Process i:

<table>
<thead>
<tr>
<th>VP 0</th>
<th>VP 1</th>
<th>VP 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>WRITE</td>
<td>Page 4</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Process j:

<table>
<thead>
<tr>
<th>VP 0</th>
<th>VP 1</th>
<th>VP 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>WRITE</td>
<td>Page 6</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Outline

- Address spaces
- VM as a tool for caching
- VM as a tool for memory management
- VM as a tool for memory protection
- Address translation

Summary of Address Translation Symbols

- Basic Parameters
  - \( N = 2^n \): Number of addresses in virtual address space
  - \( M = 2^m \): Number of addresses in physical address space
  - \( P = 2^p \): 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
  - CO: Byte offset within cache line
  - CI: Cache index
  - CT: Cache tag

Address Translation With a Page Table

- Page table entries (PTEs) are cached in L1 like any other memory word
  - PTE hit still requires a small L1 delay
  - PTEs may be evicted by other data references

  Solution: Translation Lookaside Buffer (TLB)
  - Small hardware cache in MMU
  - Contains complete page table entries for a small number of pages

Multi-Level Page Tables

- Suppose:
  - 4KB \( (2^{12}) \) page size, 48-bit address space, 8-byte PTE

- Problem:
  - Would need a 512 GB page table!
    - \( (2^{48} / 2^{12}) * 2^8 = 2^{30} * 2^8 \) bytes

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

32 bit addresses, 4KB pages, 4-byte PTEs

Level 1 page table

Level 2 page tables

Virtual memory

PTE 0
PTE 1
PTE 2 (null)
PTE 3 (null)
PTE 4 (null)
PTE 5 (null)
PTE 6 (null)
PTE 7 (null)
PTE 8

PTE 9

2K allocated VM pages for code and data

VP 0
VP 1023

VP 2047

VP 9215

2K unallocated VM pages

1MB unallocated VM pages

1 unallocated VM page for the stack

Summary

• Programmer's view of virtual memory
  • Each process has its own private linear address space
  • Cannot be corrupted by other processes

• System view of virtual memory
  • Uses memory efficiently by caching virtual memory pages
    • Efficient only because of locality
  • Simplifies memory management and programming
  • Simplifies protection by providing a convenient interpositioning point to check permissions