Today

- Virtual memory (VM)
  - Overview and motivation
  - VM as tool for caching
  - VM as tool for memory management
  - VM as tool for memory protection
  - Address translation

Virtual Memory (Previous Lectures)

- Programs refer to virtual memory addresses
  - `movl (%ecx), %eax`
  - Conceptually very large array of bytes
  - Each byte has its own address
  - Actually implemented with hierarchy of different memory types
  - System provides address space private to particular “process”

- Allocation: Compiler and run-time system
  - Where different program objects should be stored
  - All allocation within single virtual address space

- But why virtual memory?
- Why not physical memory?
Problem 1: How Does Everything Fit?

64-bit addresses: 16 Exabyte

Physical main memory: Few Gigabytes

And there are many processes ....

Problem 2: Memory Management

Physical main memory

Process 1
Process 2
Process 3
...
Process n

stack
heap
.text
data

What goes where?
Problem 3: How To Protect

Problem 4: How To Share?

Solution: Level Of Indirection

- Each process gets its own private memory space
- Solves the previous problems
Address Spaces

- **Linear address space:** Ordered set of contiguous non-negative integer addresses:
  \[ \{0, 1, 2, 3 \ldots \} \]

- **Virtual address space:** Set of \( N = 2^n \) virtual addresses
  \[ \{0, 1, 2, 3, \ldots, N-1\} \]

- **Physical address space:** Set of \( M = 2^m \) physical addresses (\( n >> m \))
  \[ \{0, 1, 2, 3, \ldots, M-1\} \]

- Clean distinction between data (bytes) and their attributes (addresses)
- Each object can now have multiple addresses
- Every byte in main memory:
  - one physical address, one (or more) virtual addresses

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 desktops, laptops, workstations
- One of the great ideas in computer science
- **MMU checks the cache**

Why Virtual Memory (VM)?

- **Efficient use of limited main memory (RAM)**
  - Use RAM as a cache for the parts of a virtual address space
    - some non-cached parts stored on disk
    - some (unallocated) non-cached parts stored nowhere
  - Keep only active areas of virtual address space in memory
    - transfer data back and forth as needed

- **Simplifies memory management for programmers**
  - Each process gets the same full, private linear address space

- **Isolates address spaces**
  - One process can’t interfere with another’s memory
    - because they operate in different address spaces
  - User process cannot access privileged information
    - different sections of address spaces have different permissions
VM as a Tool for Caching

- Virtual memory: array of \( N = 2^n \) contiguous bytes
  - think of the array (allocated part) as being stored on disk
- Physical main memory (DRAM) = cache for allocated virtual memory
- Blocks are called pages; size = \( 2^p \)

Memory Hierarchy: Core 2 Duo

L1/L2 cache: 64 B blocks

Throughput:
- 16 B/cycle
- 8 B/cycle
- 2 B/cycle
- 1 B/30 cycles

Latency:
- 3 cycles
- 14 cycles
- 100 cycles
- millions

Disk

Miss penalty (latency): 30x

Miss penalty (latency): 10,000x
DRAM Cache Organization

- DRAM cache organization driven by the enormous miss penalty
  - DRAM is about $10x$ slower than SRAM
  - Disk is about $10,000x$ slower than DRAM
    - For first byte, faster for next byte

- Consequences
  - Large page (block) size: typically 4-8 KB, sometimes 4 MB
  - Fully associative
    - Any VP can be placed in any PP
    - Requires a “large” mapping function – different from CPU caches
  - Highly sophisticated, expensive replacement algorithms
    - Too complicated and open-ended to be implemented in hardware
  - Write-back rather than write-through

Address Translation: Page Tables

- A page table is an array of page table entries (PTEs) that maps virtual pages to physical pages. Here: 8 VPs
  - Per-process kernel data structure in DRAM

![Diagram of page table and memory mapping]
Address Translation With a Page Table

Page table base register (PTBR)

Virtual address

Virtual page number (VPN) Virtual page offset (VPO)

Page table

Valid Physical page number (PPN)

Physical page number (PPN) Physical page offset (PPO)

Valid bit = 0: page not in memory (page fault)

Page Hit

- **Page hit**: reference to VM word that is in physical memory
Page Miss

- **Page miss**: reference to VM word that is not in physical memory

### Handling Page Fault

- Page miss causes page fault (an exception)
Handling Page Fault

- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here VP 4)
Handling Page Fault

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

Why does it work? Locality

- 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
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

![Diagram of virtual address space and physical address space](image)

---

VM as a Tool for Memory 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)
Simplifying Linking and Loading

■ Linking
- Each program has similar virtual address space
- Code, stack, and shared libraries always start at the same address

■ Loading
- `execve()` allocates virtual pages for `.text` and `.data` sections
  - creates PTEs marked as invalid
- The `.text` and `.data` sections are copied, page by page, on demand by the virtual memory system

VM as a Tool for Memory Protection

■ Extend PTEs with permission bits
■ Page fault handler checks these before remapping
  - If violated, send process SIGSEGV signal (segmentation fault)
Address Translation: Page Hit

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

Address Translation: Page Fault

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

- Page table entries (PTEs) are cached in L1 like any other memory word
  - PTEs may be evicted by other data references
  - PTE hit still requires a 1-cycle delay
- **Solution:** *Translation Lookaside Buffer (TLB)*
  - Small hardware cache in MMU
  - Maps virtual page numbers to physical page numbers
  - Contains complete page table entries for small number of pages

---

**TLB Hit**

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

Simple Memory System Example

- Addressing
  - 14-bit virtual addresses
  - 12-bit physical address
  - Page size = 64 bytes
Simple Memory System Page Table

Only show first 16 entries (out of 256)

<table>
<thead>
<tr>
<th>VPN</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>02</td>
<td>33</td>
<td>1</td>
</tr>
<tr>
<td>03</td>
<td>02</td>
<td>1</td>
</tr>
<tr>
<td>04</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>05</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>06</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>07</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VPN</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>09</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>0A</td>
<td>09</td>
<td>1</td>
</tr>
<tr>
<td>0B</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0C</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0D</td>
<td>2D</td>
<td>1</td>
</tr>
<tr>
<td>0E</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>0F</td>
<td>0D</td>
<td>1</td>
</tr>
</tbody>
</table>

Simple Memory System TLB

- 16 entries
- 4-way associative

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>03</td>
<td>–</td>
<td>0</td>
<td>09</td>
<td>0D</td>
<td>1</td>
<td>00</td>
<td>–</td>
<td>0</td>
<td>07</td>
<td>02</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>03</td>
<td>2D</td>
<td>1</td>
<td>02</td>
<td>–</td>
<td>0</td>
<td>04</td>
<td>–</td>
<td>0</td>
<td>0A</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>–</td>
<td>0</td>
<td>08</td>
<td>–</td>
<td>0</td>
<td>06</td>
<td>–</td>
<td>0</td>
<td>03</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>07</td>
<td>–</td>
<td>0</td>
<td>03</td>
<td>0D</td>
<td>1</td>
<td>0A</td>
<td>34</td>
<td>1</td>
<td>02</td>
<td>–</td>
<td>0</td>
</tr>
</tbody>
</table>
Simple Memory System Cache

- 16 lines, 4-byte block size
- Physically addressed
- Direct mapped

Current state of caches/tables

TLB

Cache

Page table
Address Translation Example #1

Virtual Address: 0x03D4

Physical Address

Address Translation Example #2

Virtual Address: 0x0B8F

Physical Address
Address Translation Example #3

Virtual Address: 0x0020

Physical Address

Servicing a Page Fault

1. Processor signals disk controller
   - Read block of length P starting at disk address X and store starting at memory address Y

2. Read occurs
   - Direct Memory Access (DMA)
   - Under control of I/O controller

3. Controller signals completion
   - Interrupts processor
   - OS resumes suspended process
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

Memory System Summary

- **L1/L2 Memory Cache**
  - Purely a speed-up technique
  - Behavior invisible to application programmer and (mostly) OS
  - Implemented totally in hardware

- **Virtual Memory**
  - Supports many OS-related functions
    - Process creation, task switching, protection
  - Software
    - Allocates/shares physical memory among processes
    - Maintains high-level tables tracking memory type, source, sharing
    - Handles exceptions, fills in hardware-defined mapping tables
  - Hardware
    - Translates virtual addresses via mapping tables, enforcing permissions
    - Accelerates mapping via translation cache (TLB)