Virtual Memory

Today

- Motivations for VM
- Address translation
- Accelerating translation with TLBs
A system with physical memory only

Addresses generated by the CPU correspond directly to bytes in physical memory

E.g. most Cray machines, early PCs, nearly all embedded systems, etc.
A system with virtual memory

Modern processors use virtual addresses

Hardware converts virtual addresses to physical addresses via OS-managed page table

E.g. workstations, servers, modern PCs, etc.
Motivations for virtual memory

- Use physical DRAM as a cache for the disk
  - Address space of a process can exceed physical memory size
  - Sum of address spaces of multiple processes can exceed physical memory

- Simplify memory management
  - Multiple processes resident in main memory.
    - Each process with its own address space
  - Only “active” code and data is actually in memory
    - Allocate more memory to process as needed.

- Provide protection
  - One process can’t interfere with another.
    - because they operate in different address spaces.
  - User process cannot access privileged information
    - different sections of address spaces have different permissions.
Motivation #1: DRAM a “cache” for disk

- Full address space is quite large:
  - 32-bit addresses: $\sim 4,000,000,000$ (4 billion) bytes
  - 64-bit addresses: $\sim 16,000,000,000,000,000,000,000$ (16 quintillion) bytes

- Disk storage is $\sim 300X$ cheaper than DRAM storage
  - 80 GB of DRAM: $\sim$ $33,000$
  - 80 GB of disk: $\sim$ $110$

- To access large amounts of data in a cost-effective manner, the bulk of the data must be stored on disk
### Levels in memory hierarchy

<table>
<thead>
<tr>
<th>Storage Level</th>
<th>Register Size</th>
<th>Speed</th>
<th>Cost/$Mbyte</th>
<th>Line Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Registers</td>
<td>32 B</td>
<td>1 ns</td>
<td>$125/MB</td>
<td>8 B</td>
</tr>
<tr>
<td>Cache</td>
<td>32 KB-4 MB</td>
<td>2 ns</td>
<td>$0.20/MB</td>
<td>32 B</td>
</tr>
<tr>
<td>Memory</td>
<td>1024 MB</td>
<td>30 ns</td>
<td>$0.001/MB</td>
<td>4 KB</td>
</tr>
<tr>
<td>Disk Memory</td>
<td>100 GB</td>
<td>8 ms</td>
<td>$0.001/MB</td>
<td></td>
</tr>
</tbody>
</table>

- Larger, slower, cheaper
DRAM vs. SRAM as a “cache”

- DRAM vs. disk is more extreme than SRAM vs. DRAM
  - Access latencies:
    - DRAM \(~10\)X slower than SRAM
    - Disk \(~100,000\)X slower than DRAM
  - Importance of exploiting spatial locality:
    - First byte is \(~100,000\)X slower than successive bytes on disk
      - vs. \(~4\)X improvement for page-mode vs. regular accesses to DRAM
  - Bottom line:
    - Design decisions made for DRAM caches driven by enormous cost of misses
Impact of properties on design

If DRAM was to be organized similar to an SRAM cache, how would we set the following design parameters?

- Line size? Large, since disk better at transferring large blocks
- Associativity? High, to minimize miss rate
- Write through or write back?
  - Write back, since can’t afford to perform small writes to disk

What would the impact of these choices be on:

- Miss rate: Extremely low. << 1%
- Hit time: Must match cache/DRAM performance
- Miss latency: Very high. ~20ms
- Tag storage overhead: Low, relative to block size
Locating an object in a “Cache”

- SRAM Cache
  - Tag stored with cache line
  - Maps from cache block to memory blocks
    - From cached to uncached form
    - Save a few bits by only storing tag
  - No tag for block not in cache
  - Hardware retrieves information
    - Can quickly match against multiple tags

---

Object Name

\[ X \]

= \( X \)?
Locating an object in "Cache" (cont.)

- **DRAM Cache**
  - Each allocated page of virtual memory has entry in *page table*
  - Mapping from virtual pages to physical pages
    - From uncached form to cached form
  - Page table entry even if page not in memory
    - Specifies disk address
    - Only way to indicate where to find page
  - OS retrieves information

<table>
<thead>
<tr>
<th>Object Name</th>
<th>Location</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>D: 0</td>
<td>0: 243</td>
</tr>
<tr>
<td></td>
<td>J: On Disk</td>
<td>1: 17</td>
</tr>
<tr>
<td></td>
<td>X: 1</td>
<td>N-1: 105</td>
</tr>
</tbody>
</table>
Page faults (like “cache misses”)

- What if an object is on disk rather than in memory?
  - Page table entry indicates virtual address not in memory
  - OS exception handler invoked to move data from disk into memory
    - current process suspends, others can resume
    - OS has full control over placement, etc.
Servicing a page fault

- Processor signals controller
  - Read block of length \( P \) starting at disk address \( X \) and store starting at memory address \( Y \)

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

- I/O controller signals completion
  - Interrupt processor
  - OS resumes suspended process
Motivation #2: Memory management

- Multiple processes can reside in physical memory.
- How do we resolve address conflicts?
  - what if two processes access something at the same address?

Linux/x86 process memory image
Solution: Separate virtual addr. spaces

- Virtual and physical address spaces divided into equal-sized blocks
  - blocks are called “pages” (both virtual and physical)
- Each process has its own virtual address space
  - operating system controls how virtual pages are assigned to physical memory

Virtual Address Space for Process 1:

- VP 1
- VP 2
- ...
- N-1

Virtual Address Space for Process 2:

- VP 1
- VP 2
- ...
- N-1

Address Translation:

- 0
- N-1

Physical Address Space (DRAM):

- PP 2
- PP 7
- PP 10
- M-1

(e.g., read/only library code)
Motivation #3: Protection

Page table entry contains access rights information. Hardware enforces this protection. Trap into OS if violation occurs.
VM address translation

- Virtual Address Space
  - \( V = \{0, 1, \ldots, N-1\} \)

- Physical Address Space
  - \( P = \{0, 1, \ldots, M-1\} \)
  - \( M < N \)

- Address Translation
  - \( MAP: \ V \rightarrow \ P \cup \{\emptyset\} \)
  - For virtual address \( a \):
    - \( MAP(a) = a' \) if data at virtual address \( a \) is at physical address \( a' \) in \( P \)
    - \( MAP(a) = \emptyset \) if data at virtual address \( a \) is not in physical memory
      - Either invalid or stored on disk
VM address translation: Miss

Processor

\( a \)

Hardware Addr Trans Mechanism

\( a' \)

Main Memory

virtual address

part of the on-chip memory mgmt unit (MMU)

physical address
VM address translation: Miss

Processor \[a\] virtual address

Hardware Addr Trans Mechanism part of the on-chip memory mgmt unit (MMU)

fault handler page fault

Main Memory

Secondary memory

\[a\] physical address

OS performs this transfer (only if miss)
VM address translation

- Parameters
  - \( P = 2^p \) = page size (bytes).
  - \( N = 2^n \) = Virtual address limit
  - \( M = 2^m \) = Physical address limit

Page offset bits don’t change as a result of translation
Page tables

Virtual Page Number

Memory resident page table (physical page or disk address)

Physical Memory

Disk Storage (swap file or regular file system file)
Address translation via page table

page table base register

virtual address

VPN acts as table index

valid access physical page number (PPN)

if valid=0 then page not in memory

physical address

EECS 213 Introduction to Computer Systems
Northwestern University

Monday, November 14, 2011
Page table operation

- Translation
  - Separate (set of) page table(s) per process
  - VPN forms index into page table (points to a page table entry)
Page table operation

- Computing physical address
  - Page Table Entry (PTE) provides info about page
    - if (valid bit = 1) then the page is in memory.
      - Use physical page number (PPN) to construct address
    - if (valid bit = 0) then the page is on disk - page fault
Page table operation

- Checking protection
  - Access rights field indicate allowable access
    - e.g., read-only, read-write, execute-only
    - typically support multiple protection modes
  - Protection violation fault if user doesn’t have necessary permission
Checkpoint
Multi-level page tables

- **Given:**
  - 4KB \(2^{12}\) page size
  - 32-bit address space
  - 4-byte PTE

- **Problem:**
  - Would need a 4 MB page table!
    - \(2^{20} \times 4\) bytes

- **Common solution**
  - multi-level page tables
  - e.g., 2-level table (P6)
    - Level 1 table: 1024 entries, each of which points to a Level 2 page table.
    - Level 2 table: 1024 entries, each of which points to a page
Integrating VM and cache

- Most caches accessed by physical addresses
  - Allows multiple processes to have blocks in cache at a time
  - Allows multiple processes to share pages
  - Cache doesn’t need to be concerned with protection issues
    - Access rights checked as part of address translation

- Perform address translation before cache lookup
  - But this could involve a memory access itself (of the PTE)
  - Of course, page table entries can also become cached.

![Diagram of CPU, translation, cache, and main memory with VA, PA, data, hit, miss paths]
Speeding up translation with a TLB

- "Translation Lookaside Buffer" (TLB)
  - Small hardware cache with high associativity in MMU
  - Maps virtual page numbers to physical page numbers
  - Contains complete page table entries for small number of pages
Address translation with a TLB

- **Virtual Address**: 
  - `n-1` virtual page number
  - `p` page offset
  - `0` virtual address

- **Physical Address**: 
  - valid
  - tag
  - physical page number

- **TLB Hit**: 
  - valid
  - tag
  - data

- **Cache Hit**: 
  - valid
  - tag
  - data

- **Cache**: 
  - index
  - byte offset

- **TLB**: 
  - Tag
  - Offset

- **Diagram Notes**: 
  - TLB Hit
  - Cache Hit
  - Cache
**Taken stock – main themes**

- **Programmer’s view**
  - Large “flat” address space
    - Can allocate large blocks of contiguous addresses
  - Processor “owns” machine
    - Has private address space
    - Unaffected by behavior of other processes

- **System view**
  - Virtual address space created by mapping to set of pages
    - Need not be contiguous
    - Allocated dynamically
    - Enforce protection during address translation
  - OS manages many processes simultaneously
    - Continually switching among processes
    - Especially when one must wait for resource
      - E.g., disk I/O to handle page fault
Simple memory system

- Memory is byte addressable
- Access are to 1-byte words
- 14-bit virtual addresses, 12-bit physical address
  - Page size = 64 bytes ($2^6$)
### Simple memory system page table

- Only show first 16 entries

<table>
<thead>
<tr>
<th>VPN</th>
<th>PPN</th>
<th>Valid</th>
<th>VPN</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>28</td>
<td>1</td>
<td>08</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>–</td>
<td>0</td>
<td>09</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>02</td>
<td>33</td>
<td>1</td>
<td>0A</td>
<td>09</td>
<td>1</td>
</tr>
<tr>
<td>03</td>
<td>02</td>
<td>1</td>
<td>0B</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>04</td>
<td>–</td>
<td>0</td>
<td>0C</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>05</td>
<td>16</td>
<td>1</td>
<td>0D</td>
<td>2D</td>
<td>1</td>
</tr>
<tr>
<td>06</td>
<td>–</td>
<td>0</td>
<td>0E</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>07</td>
<td>–</td>
<td>0</td>
<td>0F</td>
<td>0D</td>
<td>1</td>
</tr>
</tbody>
</table>
**Simple memory system TLB**

- **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>

**VPN**

**VPO**

**TLBT**

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

<table>
<thead>
<tr>
<th>Idx</th>
<th>Tag</th>
<th>Valid</th>
<th>B0</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>Idx</th>
<th>Tag</th>
<th>Valid</th>
<th>B0</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>19</td>
<td>1</td>
<td>99</td>
<td>11</td>
<td>23</td>
<td>11</td>
<td>8</td>
<td>24</td>
<td>1</td>
<td>3A</td>
<td>00</td>
<td>51</td>
<td>89</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
<td>0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>9</td>
<td>2D</td>
<td>0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>2</td>
<td>1B</td>
<td>1</td>
<td>00</td>
<td>02</td>
<td>04</td>
<td>08</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>36</td>
<td>0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>A</td>
<td>2D</td>
<td>1</td>
<td>93</td>
<td>15</td>
<td>DA</td>
<td>3B</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>1</td>
<td>43</td>
<td>6D</td>
<td>8F</td>
<td>09</td>
<td>B</td>
<td>0B</td>
<td>0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>5</td>
<td>0D</td>
<td>1</td>
<td>36</td>
<td>72</td>
<td>F0</td>
<td>1D</td>
<td>C</td>
<td>12</td>
<td>0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>6</td>
<td>31</td>
<td>0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>D</td>
<td>16</td>
<td>1</td>
<td>04</td>
<td>96</td>
<td>34</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>1</td>
<td>11</td>
<td>C2</td>
<td>DF</td>
<td>03</td>
<td>E</td>
<td>13</td>
<td>1</td>
<td>83</td>
<td>77</td>
<td>1B</td>
<td>D3</td>
</tr>
</tbody>
</table>
Simple memory system cache

- Cache
  - 16 lines
  - 4-byte line size
  - Direct mapped

```
<table>
<thead>
<tr>
<th>Idx</th>
<th>Tag</th>
<th>Valid</th>
<th>B0</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>19</td>
<td>1</td>
<td>99</td>
<td>11</td>
<td>23</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>1B</td>
<td>1</td>
<td>00</td>
<td>02</td>
<td>04</td>
<td>08</td>
</tr>
<tr>
<td>3</td>
<td>36</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>1</td>
<td>43</td>
<td>6D</td>
<td>8F</td>
<td>09</td>
</tr>
<tr>
<td>5</td>
<td>0D</td>
<td>1</td>
<td>36</td>
<td>72</td>
<td>F0</td>
<td>1D</td>
</tr>
<tr>
<td>6</td>
<td>31</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>1</td>
<td>11</td>
<td>C2</td>
<td>DF</td>
<td>03</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Idx</th>
<th>Tag</th>
<th>Valid</th>
<th>B0</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>24</td>
<td>1</td>
<td>3A</td>
<td>00</td>
<td>51</td>
<td>89</td>
</tr>
<tr>
<td>9</td>
<td>2D</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>A</td>
<td>2D</td>
<td>1</td>
<td>93</td>
<td>15</td>
<td>DA</td>
<td>3B</td>
</tr>
<tr>
<td>B</td>
<td>0B</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>C</td>
<td>12</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>D</td>
<td>16</td>
<td>1</td>
<td>04</td>
<td>96</td>
<td>34</td>
<td>15</td>
</tr>
<tr>
<td>E</td>
<td>13</td>
<td>1</td>
<td>83</td>
<td>77</td>
<td>1B</td>
<td>D3</td>
</tr>
<tr>
<td>F</td>
<td>14</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
```
Checkpoint
Harsh reality

- **Memory matters**
- **Memory is not unbounded**
  - It must be allocated and managed
  - Many applications are memory dominated
    - Especially those based on complex, graph algorithms
- **Memory referencing bugs especially pernicious**
  - Effects are distant in both time and space
- **Memory performance is not uniform**
  - Cache and virtual memory effects can greatly affect program performance
  - Adapting program to characteristics of memory system can lead to major speed improvements