Introduction to Computer Systems
15-213/18-243, spring 2009
17th Lecture, Mar. 19th

Instructors:
Gregory Kesden and Markus Püschel
Each process gets its own private memory space
Solves the previous problems
A System Using Virtual Addressing

- Used in all modern desktops, laptops, workstations
- One of the great ideas in computer science
- **MMU checks the cache**
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
  - Allocation, multi-level page tables

- Linux VM system
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
Latency: 3 cycles

Miss penalty (latency): 30x

L1 L-cache
32 KB

L1 D-cache

Miss penalty (latency): 10,000x

L2 unified cache
~4 MB

Main Memory
~4 GB

1 B/30 cycles
millions

Disk
~500 GB
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
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
Address Translation With a Page Table

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)

Page table base register (PTBR)

Page table address for process

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)

<table>
<thead>
<tr>
<th>Virtual address</th>
<th>Physical page number or disk address</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTE 0</td>
<td>Valid: 0/1, null</td>
</tr>
<tr>
<td></td>
<td>0/1/1/0</td>
</tr>
<tr>
<td></td>
<td>1/0/1/0</td>
</tr>
<tr>
<td>PTE 7</td>
<td>Valid: 0/1, null</td>
</tr>
<tr>
<td></td>
<td>0/1/1/0</td>
</tr>
<tr>
<td></td>
<td>1/0/1/0</td>
</tr>
</tbody>
</table>

Physical memory (DRAM)
- PP 0
- PP 3
- VP 1
- VP 2
- VP 7
- VP 4

Virtual memory (disk)
- VP 1
- VP 2
- VP 3
- VP 4
- VP 6
- VP 7

Memory resident page table (DRAM)
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
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
  - Allocation, multi-level page tables

- Linux VM system
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

Virtual Address Space for Process 1:

Virtual Address Space for Process 2:

Address translation

Physical Address Space (DRAM)
(e.g., read-only library code)
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)

![Address translation diagram]

**Virtual Address Space for Process 1:**

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

**Physical Address Space (DRAM):**

- PP 2
- PP 6
- PP 8
- ...
- M-1

**Virtual Address Space for Process 2:**

- VP 1
- VP 2
- ...
- N-1
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
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
  - Allocation, multi-level page tables

- **Linux VM 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 (segmentation fault)
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
  - Allocation, multi-level page tables

- Linux VM system
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
A TLB hit eliminates a memory access
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

```
13 12 11 10 9 8 7 6 5 4 3 2 1 0

VPN                     VPO
Virtual Page Number     Virtual Page Offset
```

```
11 10 9 8 7 6 5 4 3 2 1 0

PPN                     PPO
Physical Page Number    Physical Page Offset
```
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

**Diagram:**

```
Set | Tag | PPN | Valid | Tag | PPN | Valid | Tag | PPN | Valid | Tag | PPN | Valid
--- | --- | --- | ----- | --- | --- | ----- | --- | --- | ----- | --- | --- | -----
0   | 03  | -   | 0     | 09  | 0D  | 1     | 00  | -   | 0     | 07  | 02  | 1
1   | 03  | 2D  | 1     | 02  | -   | 0     | 04  | -   | 0     | 0A  | -   | 0
2   | 02  | -   | 0     | 08  | -   | 0     | 06  | -   | 0     | 03  | -   | 0
3   | 07  | -   | 0     | 03  | 0D  | 1     | 0A  | 34  | 1     | 02  | -   | 0
```
## Simple Memory System Cache

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

![Diagram of memory system cache with CT, CI, CO, PPN, and PPO labels]

<table>
<thead>
<tr>
<th>PPO</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>CO</td>
<td>CI</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

### Table: Cache States

<table>
<thead>
<tr>
<th>Index</th>
<th>Tag</th>
<th>Valid</th>
<th>Block 0</th>
<th>Block 1</th>
<th>Block 2</th>
<th>Block 3</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>Index</th>
<th>Tag</th>
<th>Valid</th>
<th>Block 0</th>
<th>Block 1</th>
<th>Block 2</th>
<th>Block 3</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>
Address Translation Example #1

Virtual Address: 0x03D4

Physical Address

CO | 0  | Cl | 0x5  | CT | 0x0D  | Hit? | Y  | Byte: | 0x36
Address Translation Example #2

Virtual Address: 0x0B8F

Physical Address
Address Translation Example #3

Virtual Address: 0x0020

<table>
<thead>
<tr>
<th>TLBT</th>
<th>TLBI</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

VPN 0x00  TLBI 0  TLBT 0x00  TLB Hit? N  Page Fault? N  PPN: 0x28

Physical Address

<table>
<thead>
<tr>
<th>CT</th>
<th>CI</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

PPN 0  PPO |

CO 0  CI 0x8  CT 0x28  Hit? N  Byte: Mem
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
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
  - Allocation, multi-level page tables

- Linux VM system
Allocating Virtual Pages

Example: Allocating VP5

Physical page number or disk address

Memory resident page table (DRAM)

Physical memory (DRAM)

Virtual memory (disk)
Allocating Virtual Pages

- Example: Allocating VP 5
- Kernel allocates VP 5 on disk and points PTE 5 to it
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
A Two-Level Page Table Hierarchy

Level 1 page table

Level 2 page tables

Virtual memory

2K allocated VM pages for code and data

6K unallocated VM pages

1023 unallocated pages

1 allocated VM page for the stack

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
(1K - 9) null PTEs

PTE 0
PTE 1
PTE 2 (null)
PTE 3 (null)
PTE 4 (null)
PTE 5 (null)
PTE 6 (null)
PTE 7 (null)
PTE 8
(1K - 9) null PTEs

1023 null PTEs
PTE 1023

PTE 0
PTE 1
PTE 2 (null)
PTE 3 (null)
PTE 4 (null)
PTE 5 (null)
PTE 6 (null)
PTE 7 (null)
PTE 8
(1K - 9) null PTEs

1023 null PTEs
PTE 1023

VP 0
VP 1023
VP 1024
VP 2047
Gap
VP 9215

2K allocated VM pages for code and data

6K unallocated VM pages

1023 unallocated pages

1 allocated VM page for the stack
Translating with a k-level Page Table

Virtual Address

Physical Address
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
  - Allocation, multi-level page tables

- Linux VM system