Virtual Memory: Systems

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

Simple Memory System Example

Addressing
- 14-bit virtual addresses
- 12-bit physical address
- Page size = 64 bytes

Review of 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 the virtual address (VA)
- VPN: Virtual page number
- VPO: Virtual page offset
- TLBI: TLB index
- TLBT: TLB tag

Components of the physical address (PA)
- PPN: Physical page number
- PPO: Physical page offset (same as VPO)
- CT: Cache tag
- CI: Cache index
- CO: Byte offset within cache line

1. Simple Memory System TLB

16 entries
4-way associative
2. 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>10</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>02</td>
<td>03</td>
<td>1</td>
</tr>
<tr>
<td>03</td>
<td>02</td>
<td>1</td>
</tr>
<tr>
<td>04</td>
<td>03</td>
<td>0</td>
</tr>
<tr>
<td>05</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>06</td>
<td>06</td>
<td>0</td>
</tr>
<tr>
<td>07</td>
<td>07</td>
<td>0</td>
</tr>
</tbody>
</table>

3. Simple Memory System Cache

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

<table>
<thead>
<tr>
<th>TLB</th>
<th>TLBI</th>
<th>TLBT</th>
<th>Hit?</th>
<th>Page Fault?</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
</tbody>
</table>

Address Translation Example #1

Virtual Address: 0x003D4

Physical Address

1. Simple Memory System TLB

16 entries
4-way associative
3. Simple Memory System Cache

16 lines, 4-byte block size

Physically addressed

Direct mapped

Virtual Address: 0x03D4

Physical Address: 0x03D4

Address Translation Example #1

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

VPN TLB

VPN TLBI TLBT TLB Hit? Page Fault? PPN

CO CI CT Hit? Byte:

Address Translation Example #1

Virtual Address: 0x03D4

Physical Address: 0x03D4

Address Translation Example #1

Virtual Address: 0x03D4

Physical Address: 0x03D4

Address Translation Example #1

Virtual Address: 0x03D4

Physical Address: 0x03D4
Address Translation Example #2

Virtual Address: 0x0020

Physical Address

1. Simple Memory System TLB
16 entries
4-way associative

2. Simple Memory System Page Table
Only show first 16 entries (out of 256)
Address Translation Example #2

Virtual Address: 0x0020

Physical Address

3. Simple Memory System Cache

16 lines, 4-byte block size
Physically addressed
Direct mapped
Address Translation Example #3

Virtual Address: 0x0316

<table>
<thead>
<tr>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VPN</td>
<td>TLB</td>
<td>TLBI</td>
<td>TLBT</td>
<td>VPN</td>
<td>TLB</td>
<td>TLBI</td>
<td>TLBT</td>
<td>Page Fault?</td>
<td>PPN</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Physical Address

<table>
<thead>
<tr>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CO</td>
<td>CI</td>
<td>CT</td>
<td>Hit?</td>
<td>Byte</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Physical Address

<table>
<thead>
<tr>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CO</td>
<td>CI</td>
<td>CT</td>
<td>Hit?</td>
<td>Byte</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Simple Memory System TLB

16 entries
4-way associative

<table>
<thead>
<tr>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLB</td>
<td>TLBI</td>
<td>TLBT</td>
<td>VPN</td>
<td>PPN</td>
<td>Valid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Simple Memory System Page Table

Only show first 16 entries (out of 256)

<table>
<thead>
<tr>
<th>00</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VPN</td>
<td>PPN</td>
<td>Valid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
</tr>
<tr>
<td>13</td>
<td>17</td>
<td>01</td>
<td>01</td>
<td>01</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VPN</td>
<td>PPN</td>
<td>Valid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Address Translation Example #3

Virtual Address: 0x0316

Physical Address

Intel Core i7 Memory System

End-to-End Core i7 Address Translation
Core i7 Level 1-3 Page Table Entries

<table>
<thead>
<tr>
<th>KB</th>
<th>12</th>
<th>9</th>
<th>6</th>
<th>3</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>XD</td>
<td>Unused</td>
<td>Page table physical base address</td>
<td>Unused</td>
<td>G</td>
<td>P5</td>
</tr>
</tbody>
</table>

Available for OS (page table locations only)

Each entry references a 4K child page table. Significant fields:
- 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/N: User or supervisor (kernel) mode access permission for all reachable pages.
- WT: Write-through or write-back cache policy 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 2 PTEs only).
- Page table base address: 40 most significant bits of physical page table address (forces page tables to be 4KB aligned).
- XD: Disable or enable instruction fetches from all pages reachable from this PTE.

---

Core i7 Level 4 Page Table Entries

<table>
<thead>
<tr>
<th>12</th>
<th>9</th>
<th>6</th>
<th>3</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>XD</td>
<td>Unused</td>
<td>Page physical base address</td>
<td>Unused</td>
<td>G</td>
</tr>
</tbody>
</table>

Available to OS (page locations only)

Each entry references a 4K child page. Significant fields:
- P: Child page is present in memory (1) or not (0).
- R/W: Read-only or read-write access permission for child page
- U/N: User or supervisor mode access
- WT: Write-through or write-back cache policy for this page
- A: Reference bit (set by MMU on reads and writes, cleared by software)
- D: Dirty bit (set by MMU on writes, cleared by software)
- Page physical base address: 40 most significant bits of physical page address (forces pages to be 4KB aligned)
- XD: Disable or enable instruction fetches from this page.

---

Core i7 Page Table Translation

Virtual address

Offset into physical and virtual page

Physical address

CR3: Physical address of L1 PT

L1 PT Page global directory

L2 PT Page upper directory

L3 PT Page middle directory

L4 PT Page table

PPN

 physical address

Physical address of page

---

Cute Trick for Speeding Up L1 Access

Observation:
- Bits that determine CI are identical in virtual and physical address
- Can index into cache while address translation 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
Linux Organizes VM As Collection of “Areas”

- `task_struct`
- `mm_struct`
- `vm_area_struct`

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

Linux Page-Fault Handling

- Segmentation fault: accessing a nonexistent page
- Normal page fault
- Protection exception: e.g., violating permission by writing to a read-only page (Linux reports as Segmentation fault)

Memory Mapping

- VM areas initialized by associating them with disk objects.
  - Process is known as memory mapping.
- Area can be backed by (i.e., get its initial values from):
  - Regular file on disk (e.g., an executable object file)
  - Initial page bytes come from a section of a file
  - Anonymous file (e.g., nothing)
  - First fault will allocate physical page full of 0's (demand-zero page)
  - Once the page is written to (dirtied), it is like any other page

Dirty pages are copied back and forth between memory and a special swap file.

Sharing Revisited: Shared Objects

- Process 1 maps the shared object
Sharing Revisited: Shared Objects

- Process 2 maps the shared object
- Notice how the virtual addresses can be different.

Sharing Revisited: 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

The `fork` Function Revisited

VM and memory mapping explain how `fork` provides private address space for each process

To create virtual address for new new process
- Create exact copies of current kernel structures and page tables.
- Flag each page in both processes as read-only, private COW

On return, each process has exact copy of virtual memory
Subsequent writes create new pages using COW mechanism.
User-Level Memory Mapping

```c
void *mmap(void *start, int len,
int prot, int flags, int fd, int offset
```

Maps `len` bytes starting at offset `offset` of the file specified by file descriptor `fd`, preferably at address `start`
- `start`: may be NULL for “pick an address”
- `prot`: PROT_READ, PROT_WRITE, ...
- `flags`: MAP_ANON, MAP_PRIVATE, MAP_SHARED, ...

Returns pointer to start of mapped area (might not be `start`)

Example: Using `mmap` to Copy Files

- Copying a file to stdout without transferring data to user space.

```c
#include "mmap.h"

void mmapcopy(int fd)
{
    // File to memory mapped area
    int *m = mmap(0, 1024, PROT_READ, MAP_PRIVATE, fd, 0)
    // Write data to memory
    strcpy(m, "Hello, world!")
    // Copy data to stdout
    printf("%s", m)
}```