Virtual Memory: Systems

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

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

Simple Memory System Example

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

1. Simple Memory System TLB

16 entries
4-way associative

TLB

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>PPN</th>
<th>In Use</th>
<th>PPO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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>00</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>63</td>
<td>1</td>
</tr>
<tr>
<td>04</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>05</td>
<td>18</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>

Address Translation Example #1

Virtual Address: 0x03D4

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

1. 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>00</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>63</td>
<td>1</td>
</tr>
<tr>
<td>04</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>05</td>
<td>18</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>

Address Translation Example #1

Virtual Address: 0x03D4

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
Address Translation Example #1

Virtual Address: 0x03D4

Physical Address

VPN

TLBI

TLBT

Page Fault? N

PPN: 0x00D

CO

CI

CT

Hit?

Byte: 0x36

3. Simple Memory System Cache

16 lines, 4-byte block size

Physically addressed

Direct mapped

VPN ____ TLBI ___ TLBT ____ TLB Hit? __ Page Fault? __        PPN: ____

CO ___ CI___ CT ____ Hit? __              Byte: ____
Address Translation Example #2

Virtual Address: 0x0020

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

VPN  TLBT  TLBI  VPO

VPN[0x00]  TLBI[0]  TLBT[0]  TLB Hit?  Page Fault?  PPN:

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>
</table>
| CO | CI | CT | Hit?| Byte:

1. Simple Memory System TLB

16 entries
4-way associative

Physical Address

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

VPN  TLBT  TLBI  VPO

VPN[0x00]  TLBI[0]  TLBT[0]  TLB Hit?  Page Fault?  PPN:

Address Translation Example #2

Virtual Address: 0x0020

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

VPN  TLBT  TLBI  VPO

VPN[0x00]  TLBI[0]  TLBT[0]  TLB Hit?  Page Fault?  PPN:

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>
</table>
| CO | CI | CT | Hit?| Byte:

2. Simple Memory System Page Table

Only show first 16 entries (out of 256)
Address Translation Example #2

Virtual Address: 0x0020

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

VPN 00011211101101010101010000

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>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

VPN 00011211101101010101010000

3. Simple Memory System Cache

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

Address Translation Example #2

Virtual Address: 0x0020

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

VPN 00011211101101010101010000

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>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

VPN 00011211101101010101010000

Address Translation Example #2

Virtual Address: 0x0020

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

VPN 00011211101101010101010000

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>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

VPN 00011211101101010101010000

3. Simple Memory System Cache

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

Virtual Address: 0x0316

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 #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

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.
- 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 1 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

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.
- W: 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

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”**

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

**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*.

**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)

**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.

- Instruction writing to private page triggers protection fault
- Handler creates new R/W page
- Instruction restarts upon handler return
- Copying deferred as long as possible!

To create virtual address for 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

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, ...
- 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.